Android基础开发与实践

吴善财 编著

- 采用独特、易懂的讲解方式
- 汇集实用、易学的操作案例
- 兼顾理论、案例的完美展现





清华大学出版社

Android 基础开发与实践

吴善财　编著

清华大学出版社
北　京

内 容 简 介

本书用通俗易懂的语言，循序渐进地讲解了 Android 的各种基本知识，通过理论加实践的方式讲解了 Android 技术在各个领域的具体应用。全书共分为 20 章，其中第 1～3 章是“基础篇”，讲解 Android 的发展前景、搭建开发环境和 Android SDK 的知识；第 4～9 章是“核心技术篇”，详细讲解 Android 体系结构、UI 布局、控件、数据存储和 GPS 定位等知识；第 10～13 章是“实践闯关篇”，详细讲解 Android 在常见领域中的具体应用流程；第 14～16 章是“提高篇”，详细讲解程序优化、Graphics 编程和三维开发方面的知识；第 17～20 章是“综合实战篇”，讲解 Android 使用 Google 技术的知识，并通过 3 个综合实例的实现过程，讲解大、中型 Android 项目的开发流程。本书风格独特、内容新颖、知识全面，全书内容采用理论加实践的教学方法，阅读轻松，引人入胜。另外，还配有一张 DVD 光盘，为读者提供书中案例的源代码和视频讲解，帮助读者快速学会书中内容，掌握 Android 开发技术。

本书定位于 Android 的初、中级用户，既可以作初学者的教材，也可以作为准备向该领域发展的程序员的参考用书。

图书在版编目(CIP)数据

Android 基础开发与实践/吴善财编著. --北京：清华大学出版社，2012.8
ISBN 978-7-302-28952-4

Ⅰ. ①A…　Ⅱ. ①吴…　Ⅲ. ①移动终端—应用程序—程序设计　Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字(2012)第 111544 号

责任编辑：魏　莹
装帧设计：杨玉兰
责任校对：李玉萍
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　　邮　　编：100084
社 总 机：010-62770175　　　邮　　购：010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn, 010-62791865
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　张：**37　　**字　数：**897 千字
附 DVD 1 张
版　　次：2012 年 8 月第 1 版　　**印　次：**2012 年 8 月第 1 次印刷
印　　数：1～4000
定　　价：69.80 元

产品编号：



前　　言

进入 21 世纪以来，整个社会已经逐渐变得陌生了！生活和工作的快节奏令我们目不暇接，各种各样的信息充斥着我们的视野、撞击着我们的思维。追忆过去，Windows 操作系统的诞生成就了微软的霸主地位，也造就了 PC 时代的繁荣。然而，以 Android 和 iPhone 手机为代表的智能移动设备的发明却敲响了 PC 时代的丧钟！移动互联网时代(3G 时代)已经来临，谁会成为这些移动设备上的主宰？毫无疑问，它就是 PC 时代的 Windows——Android!

Android 魅力不可阻挡

随着 3G 技术的普及，手机网络的速率越来越快，这使得更多内容丰富的应用程序安装在手机上成为可能，例如视频通话、视频点播、手机上网、手机 QQ 聊天等。为了实现上述应用需求，必须有一个好的开发平台来支持。Google 公司积极发展，所发起的 OHA 联盟走在了业界的前列，2007 年 11 月推出了开放的 Android 平台，由于其开放性，Android 平台得到了业界广泛的支持，其中包括各大手机厂商和著名的移动运营商等。继 2008 年 9 月第一款基于 Android 平台的手机 G1 发布之后，随后三星、摩托罗拉、索爱、LG、华为等公司都将推出自 Gflg~Android 平台的手机，中国移动也将联合各手机厂商共同推出基于 Android 平台的 OPhone。

Android 操作系统是 Google 最具杀伤力的武器之一。苹果以其天才的创新，使得 iPhone 在全球迅速拥有了数百万忠实“粉丝”，而 Android 作为第一个完整、开放、免费的手机平台，使开发者在为其开发程序时拥有更大的自由。2011 年 4 月，Android 手机发货量超过塞班，第一次登上了智能手操作系统龙头老大的宝座。截止到本书完稿之时，Android 已经连续 1 年雄踞智能手机市场占有率第一，成为当前最受欢迎的智能手机系统。

本书的内容

本书用通俗易懂的语言，循序渐进地讲解了 Android 技术的各种基本知识，通过理论加

实践的方式讲解了 Android 技术在各个领域的具体应用过程。本书风格独特、内容新颖、知识全面。全书分为 20 个章节，第 1～3 章是“基础篇”，讲解 Android 的发展前景、搭建开发环境和 Android SDK 的核心知识；第 4～9 章是“核心技术篇”，详细讲解 Android 体系结构、UI 布局、控件、数据存储和 GPS 定位等核心知识；第 10～13 章是“实践闯关篇”，详细讲解 Android 在常见领域中的具体应用流程；第 14～16 章是“提高篇”，详细讲解程序优化、Graphics 编程和三维开发方面的知识；第 17～20 章是“综合实战篇”，讲解 Android 使用 Google 技术的知识，并通过 3 个综合实例的实现过程，讲解大、中型 Android 项目的开发流程。全书内容采用理论加实践的教学方法，实用性强，便于读者掌握。

本书特色

本书内容丰富、全面，几乎涵盖了 Android 开发的方方面面。在内容的编写上，本书具有以下特色。

(1) 独特讲解方式引人入胜，内容循序渐进

为了吸引读者眼球，作者用独特的讲解方式讲述全书内容。从最基本的 Android 背景讲起，到最后的综合实战，由浅入深地将 Android 应用技术分为了五篇：基础篇、核心技术篇、实践闯关篇、提高篇和综合实战篇。

(2) 结构合理，易学易懂

从用户的实际需要出发，科学安排知识结构，内容由浅入深、叙述清楚，并附有相应的注意事项和技巧提示，具有很强的知识性和实用性。本书条理清晰、语言简洁，可帮助读者快速掌握每个知识点；每个部分既相互连贯又自成体系，使读者既可以按照本书编排的章节顺序进行学习，也可以根据自己的需求对某一章节进行针对性的学习。

(3) 实用性强

本书彻底摒弃枯燥的理论和简单的操作，注重实用性和可操作性，将 Android 理论融合到实际的实战项目中，使读者掌握相关理论的同时，还能学习到这些理论在实践中的运用过程。

(4) 实例典型

书中的开发实例典型并具有创意，每一个实例都经过了精心挑选，体现了移动互联网应用所需的创新精神及良好的用户体验理念，这个设计思路很值得大家去思考和学习。

本书由吴善财主编。在编写过程中，得到了清华大学出版社工作人员的大力支持。笔者本人毕竟水平有限，如有纰漏和不尽如人意之处在所难免，诚请读者提出意见或建议，以便修订并使之更臻完善。

编　者

目　　录

第一篇　基　础　篇

第二篇　核心技术篇

第三篇 实践闯关篇

第四篇 提 高 篇

第五篇 综合实战篇

Android

第一篇 基 础 篇

第 1 章 说书先生谈 Android

Android 是当今最重要的手机开发平台之一，它是建立在 Java 开发平台之上，能够迅速建立手机软件的解决方案。Android 的功能十分强大，已成为当今软件行业的一股新兴力量。本章将简单介绍 Android 的发展历程和背景，让读者了解 Android 的兴起之路。

1.1 引 子

本人家住在山下张家村，靠上山砍柴为生。梦想混个一代大侠的差事，在江湖行侠仗义的同时能够扬名立万。我业余爱好是喜欢听说书先生讲江湖故事。今天同往常一样，我早早地将柴卖掉，一看太阳落山还早，就来到了镇上的茶馆，叫上一壶茶，边品茶边听书。

当今武林塞班、苹果、微软三大门派互不相让，在手机系统领域各领风骚。然而仅仅三足鼎立几年，便诞生了一个新的门派——安卓。此门派十分神秘，在天下广招门徒，并且完全免费。一时间，江湖中的年轻才俊纷纷去拜师，加入了安卓派。究竟安卓派能否发扬光大，还是昙花一现，且听我下次分解……

回家的路上我还在想着安卓，既然安卓派这么好，而且完全免费，我可以尝试加入安卓派，即使学不成也不会浪费掉多年的积蓄。真希望能学到一技之长，圆我的侠客梦。傍晚时分，躺在床上，满脑子还是安卓……

1.2 手 机 江 湖

今天我起了一个大早，争取尽快将柴卖完，多听说书先生讲江湖的故事。茶馆里依旧人山人海，今天讲的是“手机江湖”。

闯江湖，闹江湖，江湖本无情。谈起手机江湖，就不得不说最流行的智能手机。这些智能手机，大大丰富了人们的生活。在手机江湖中，只有智能手机才可以称之为“侠客”。今天我们就说一说手机江湖中的“侠客”。

1.2.1 怎样才能成为侠客

所谓智能手机(Smartphone)，是指“像个人电脑一样，具有独立的操作系统，可以由用户自行安装软件、游戏等第三方服务商提供的程序，通过此类程序来不断对手机的功能进行扩充，并可以通过移动通信网络来实现无线网络接入这样一类手机的总称”。

江湖中已经成名的智能手机有很多，比如 Symbian 操作系统的 S60 系列、Windows Mobile 操作系统的 Windows Mobile Smartphone 系列；还有的是传统 PDA 加上手机通信功能，比如 Windows Mobile 操作系统的 Windows Mobile Pocket PCPhone 系列、Palm 操作系统的 Treo 系列；也可以是其他独立类型，比如 Symbian 操作系统的 S80、UIQ，以及一些 Linux 操作系统的智能手机。然而，就新近的发展情况来看，这些智能手机的类型有相融合的趋势。智能手机有别普通带触摸屏的手机，一般普通带触摸屏的手机使用的都是生产厂商自行开发的封闭式操作系统，所能实现的功能非常有限。

怎样才能算是一个侠客呢？江湖中说法纷纭，在 2005 年手机武林大会期间，各派代表对侠客的基本要素做了一个统一的规范，只有具备如下功能的手机才能被称之为“侠客”。

- 高速度处理芯片。
- 大存储芯片和存储扩展能力。
- 面积大、标准化、可触摸的显示屏。
- 支持播放式的手机电视。
- 支持 GPS 导航。
- 操作系统必须支持新应用的安装。
- 配备大容量电池，并支持电池更换。

1.2.2 侠客的特点

侠客(智能手机)的主要特点如下。

- 具备普通手机的全部功能，能够进行正常的通话，发短信等。
- 具备无线接入互联网的能力，即需要支持 GSM 网络下的 GPRS 或者 CDMA 网络下的 CDMA 1X 或者 3G 网络。
- 具备 PDA 的功能，包括 PIM(个人信息管理)、日程记事、任务安排、多媒体应用、浏览网页等。
- 具备一个具有开放性的操作系统，在这个操作系统平台上，可以安装更多的应用程序，从而使智能手机的功能可以得到无限的扩充。
- 具有人性化的一面，可以根据个人需要扩展机器的功能。
- 功能强大，扩展性能强，支持多个第三方软件。

1.2.3　看已成名的侠客

当今手机江湖人才辈出，既有成名多年令人仰慕的大侠，也有后浪推出来的后起之秀。点评当今手机江湖，最有名的侠客当属塞班、微软、苹果、黑莓、PDA 和安卓。

1. Symbian OS(塞班系统)

Symbian 是由诺基亚、索尼爱立信、摩托罗拉、西门子等几家大型移动通讯设备商共同出资组建的一个合资公司，专门研发手机操作系统。现已被 NOKIA 全额收购。Symbian 很像是 Windows 和 Linux 的结合体，有着良好的界面，采用内核与界面分离技术，对硬件的要求比较低，支持 C++，Visual Basic 和 J2ME，兼容性较差。目前根据人机界面的不同，Symbian 体系的 UI(User Interface 用户界面)平台分为 Series 60、Series 80、Series 90、UIQ 等。Series 60 主要是给数字键盘手机用，而 Series 80 是为完整键盘所设计，Series 90 则是为触控笔方式而设计的。

2. Windows Mobile

Windows Mobile 是 Microsoft 用于 Pocket PC 和 Smartphone 的软件平台，它将熟悉的 Windows 桌面扩展到了个人设备中。Windows Mobile 是微软为手持设备推出的“移动版 Windows”，使用 Windows Mobile 操作系统的设备主要有 PPC 手机、PDA、随身音乐播放器等。Windows Mobile 操作系统有三种，分别是 Windows Mobile Standard、Windows Mobile Professional 和 Windows Mobile Classic。目前常用版本是 Windows Mobile 6.1，最新版本是 Windows Phone。生产 Windows Mobile 手机的最大厂商是台湾 HTC，其他还有东芝、惠普、Motorola(摩托罗拉)，华硕，索爱，三星，LG 等。

3. Palm

Palm 是流行的个人数字助理(PDA)的传统名字，是一种手持设置形式，也以掌上电脑而闻名。广义上，Palm 是 PDA 的一种，由 Palm 公司发明，这种 PDA 上的操作系统也称为 Palm，有时又称为 Palm OS。狭义上，Palm 指 Palm 公司生产的 PDA 产品，以区别于 SONY 公司的 Clie 和 Handspring 公司的 Visor/Treo 等其他运行 Palm 操作系统的 PDA 产品。其数据显示于一个液晶显示(LCD)屏。显著特点之一是其数据的基本输入方法——一个写入装置，叫做铁笔，能够通过点击显示器上的图标选择输入的项目。铁笔亦能用于在显示屏的表面手写，输入包括文字和数字的信息，这被称之为涂鸦。Palm Pilot 系列产品原是由一家叫“Palm Computing”的公司研发设计的，这个公司在历经两次并购后，成为 3Com 的一个事业部门，而后 Palm 公司又从 3Com 公司中独立出来成为一个公司。2009 年 2 月 11 日，Palm 公司 CEO Ed Colligan 宣布：以后将专注于 Web OS 和 Windows Mobile 的智能设备，除了 Palm Centro 会在以后和其他运营商合作时继续推出外，将不会再推出基于“Palm OS”的智能设备。这对于 Palm 粉丝们来说，实在是一个令人扼腕叹息的消息。

4. BlackBerry

黑莓(BlackBerry)是加拿大 RIM 公司推出的一种移动电子邮件系统终端，其特色是支持推动式电子邮件、手提电话、文字短信、互联网传真、网页浏览及其他无线资讯服务。黑莓最强大、最有优势的方面在于收发邮件，然而在中国，用手机收发邮件还不是很流行，所以黑莓在中国

几乎没有多大市场。

5. iPhone

iPhone 由苹果公司(Apple, Inc.)前首席执行官史蒂夫·乔布斯在 2007 年 1 月 9 日举行的 Macworld 宣布推出，2007 年 6 月 29 日在美国上市，将创新的移动电话、可触摸宽屏 iPod 以及具有桌面级电子邮件、网页浏览、搜索和地图功能的突破性因特网通信设备这三种产品完美地融为一体。iPhone 引入了基于大型多触点显示屏和领先性新软件的全新用户界面，让用户用手指即可控制 iPhone。iPhone 还开创了移动设备软件尖端功能的新纪元，重新定义了移动电话的功能。苹果公司前首席执行官史蒂夫·乔布斯说，“手指是我们与生俱来的终极定点设备，而 iPhone 利用它们创造了自鼠标以来最具创新意义的用户界面。”

6. Android

Android 一词的本义是指“机器人”，是于 2007 年 11 月 5 日宣布的基于 Linux 平台的开源手机操作系统的名称，该平台由操作系统、中间件、用户界面和应用软件组成，号称是首个为移动终端打造的真正开放和完整的移动软件。

各位软件开发才俊们，你们都纷纷加入到手机江湖中，想获取自己的一席之地。但是究竟是加入塞班呢，还是加入苹果呢？还是……这个得靠你们自己来决定。我想要说的是，安卓来势凶猛，大有三分天下之势。预知安卓的具体情况，且听我下回分解。

1.3 安卓来兮

和往常一样，我早早地来到了茶馆，静听说书先生的评书故事。

长江后浪推前浪，一代新人换旧人。前面说到安卓来势凶猛，但是为什么安卓这么强势呢？是不是有一个神秘的力量在后台支持呢？今天我就先从其出身说起。

Android 即安卓，虽然崛起较晚，但是一出则威震江湖。安卓出身于四大武林世家之一的 Linux 家族，Linux 家族向来做事低调，善于集思广益。在 2007 年 11 月 5 日这一天，Linux 家族推出了年轻一代的剑客——Android。Android 采用了 WebKit 浏览器引擎，具备触摸屏、高级图形显示和上网功能，用户能够在手机上查看电子邮件、搜索网址和观看视频节目等，同时 Android 还具有比 iPhone 等其他手机更强的搜索功能，可以说是一种融入全部 Web 应用的平台。

正是因为安卓具有的上述强大武功，在推出后的短短几个月，其名声便如雷贯耳，被手机江湖联盟破格封为侠客。在功成名就之后，由武林泰山北斗之誉的谷歌为其量身打造了一个绝世好剑 Android SDK，并在谷歌的支持下，成立了门派，开始广纳门徒。短短几年间，Android 弟子规模于公元 2010 年下半年便超越了苹果 iPhone。

1.3.1 安卓的发展背景

Android 功成名就之后，包括中国移动、摩托罗拉、高通、宏达电和 T-Mobile 在内的 30 多家技术和无线应用的领军企业组成联盟，希望通过与运营商、设备制造商、开发商和其他有关各方结成深层次的合作伙伴关系，借助建立标准化、开放式的移动电话软件平台，在移动产业内形成一个开放式的生态系统。

1.3.2　安卓的第一代产品

安卓派成立之后，于 2008 年 9 月 22 日和美国运营商 T-MobileUSA 合作，在纽约正式发布了第一款智能手机——T-Mobile G1。该款手机为台湾宏达电代工制造，是世界上第一部使用 Android 操作系统的手机，支持 WCDMA/HSPA 网络，理论下载速率 7.2Mbps，并支持 Wi-Fi。

图 1-1 所示为摩托罗拉的首款 Android 手机 CLIQ。



图 1-1　摩托罗拉的首款 Android 手机 CLIQ

1.3.3　Android 门派

Android 的研发队伍阵容强大，包括摩托罗拉、Google、HTC(宏达电子)、PHILIPS、T-Mobile、高通、魅族、三星、LG 以及中国移动在内共 34 家企业，这都是在江湖享誉盛名的大佬。这些企业都将基于该平台开发手机的新型业务，应用之间的通用性和互联性将在最大程度上得到保持。并且这些企业还成立了手机开放联盟，该联盟中的成员有如下几个。

1. 手机制造商

手机开放联盟中的制造商有中国台湾宏达国际电子(HTC)、摩托罗拉(美国最大的手机制造商)，韩国三星电子(仅次于诺基亚的全球第二大手机制造商)、韩国 LG 电子、中国移动(全球最大的移动运营商)、日本 KDDI、日本 NTT DoCoMo、美国 Sprint Nextel(美国第三大移动运营商)、意大利电信(Telecom Italia)、西班牙 Telefónica、T-Mobile(德意志电信旗下公司)。

2. 半导体公司

手机开放联盟中的半导体公司有 Audience Corp(声音处理器公司)、Broadcom Corp(无线半导体主要提供商)、英特尔(Intel 公司)、Marvell Technology Group、Nvidia(图形处理器公司)、SiRF(GPS 技术提供商)、Synaptics(手机用户界面技术公司)、德州仪器(Texas Instruments)、高通(Qualcomm)、惠普 HP(Hewlett-Packard Development Company，L.P)。

3. 软件公司

手机开放联盟中的软件公司有 Aplix、Ascender、eBay 的 Skype、Esmertec、Living Image、NMS Communications、Noser Engineering AG、Nuance Communications、PacketVideo、SkyPop、Sonix Network、TAT-The Astonishing Tribe 和 Wind River Systems。

1.4 何以一统天下

上回说到了安卓有雄厚的背景，也有联盟的支持，并且软件泰山北斗的谷歌为其提供了绝世好剑 Android SDK 作为武器。既然安卓来势汹汹，长期以来的三分天下难道就能被它一统天下吗？且看我下面的分解。

1.4.1 赏罚分明

安卓为了提高门人的开发积极性，不但为他们提供了一流的硬件设置，而且还提供了一流的软件服务。采取了振奋人心的奖励机制，定期召开比武大会，夺魁者将得到重奖。

1. 开发 Android 平台的应用

在 Android 平台上，程序员可以开发出各式各样的应用程序。Android 是通过 Java 语言开发的，只要读者具备 Java 开发基础，就能很快地上手并掌握。作为单独的 Android 程序开发，对 Java 的编程门槛并不高，即使没有编程经验的门外汉，也可以在突击学习 Java 之后掌握 Android。另外，Android 完全支持 2D、3D 和数据库，并且和浏览器实现了集成。所以通过 Android 平台，程序员可以迅速、高效地开发出绚丽多彩的应用，例如常见的工具、管理、互联网和游戏等。总之一句话：只要我们的创意存在，我们就会站在 Android 的最前沿。

2. 随时参加 Android 大赛

Google 为了吸引更多的用户使用 Android 进行开发，推出了 Android 设计大赛，并成功举办了奖金为 1000 万美元的开发者竞赛。鼓励用户创建出创意十足、实用性强的软件。这种大赛对于开发人员来说，不但能练习开发水平，并且高额的奖金也是学员们学习的动力。

3. 用 Android Market 获取收益

为了能让 Android 平台吸引更多的关注，谷歌开发了自己的 Android 手机软件下载店——Android Market，允许开发人员发布应用程序，也允许 Android 用户随意下载获取自己喜欢的程序。也就是说，Android Market 将是一个很大的软件市场，作为一个用户，你可以在这个市场上发布或下载应用程序。作为开发者，需要申请开发者账号，申请后才能将自己的程序上传到 Android Market，并且可以对自己的软件进行定价。所以说，只要你的软件程序足够吸引人，你就可以获得很好的金钱回报，从而达到学习、赚钱两不误。Android Market 地址是 http://www.Android.com/market/，界面效果如图 1-2 所示。



图 1-2　Android Market 主界面

1.4.2　前景会如何

既然安卓声势浩大，并且有手机联盟的支持，是不是前景光明呢？前景光明是肯定的，但是也不太能达到一统江湖的地步。江湖上没有绝对，只有强者。即使完成了一统，在分久必合、合久必分的江湖上，日后也会成为过眼云烟。但是安卓的智囊团队并不糊涂，他们为其打造了一个长远的发展规划，这是一个不求一统武林，只求千秋万载的规划。

与 iPhone 相似，Android 采用 WebKit 浏览器引擎，具备触摸屏、高级图形显示和上网功能，用户能够在手机上查看电子邮件、搜索网址和观看视频节目等，比 iPhone 等其他手机更强调搜索功能，界面更强大，可以说是一种融入全部 Web 应用的单一平台。

Android 的杀手锏是整个系统的开放性和服务免费。Android 是一个对第三方软件完全开放的平台，开发者在为其开发程序时拥有更大的自由度，突破了 iPhone 等只能添加为数不多的固定软件的枷锁；同时与 Windows Mobile、Symbian 等厂商不同，Android 操作系统免费向开发人员提供，这样可节省近三成的成本。

Android 项目目前正在从手机运营商、手机厂商、开发者和消费者那里获得大力支持。谷歌移动平台主管安迪·鲁宾(Andy Rubin)表示，与软件开发合作伙伴的密切接触正在进行中。从去年 11 月开始，谷歌开始向服务提供商、芯片厂商和手机销售商提供 Android 平台，并组建“开放手机联盟”，其成员超过 30 家。

1.4.3　Android 市场前景

根据数据显示，2008 年全球市场规模不到 100 万部，2009 年全球市场规模为 600 万部左右，

而据台湾某官方媒体预测，预期 2013 年 Android 全球手机市场规模将突破 2800 万部。未来 Android 市场也是“百花齐放、百家争鸣”的局面，各类设计公司均可参与。但值得我们考虑的是，Android 手机始终属于智能机范畴，智能机必须遵守的基本“江湖规则”有如下几条。

(1) 智能应用是带动市场的核心。

(2) 价格相对 Feature (功能)手机，始终要高一级。

(3) 品牌市场仍占上风。

个人认为上述第一条肯定不会被动摇，而且会越来越明显，谁智能应用越丰富、收费越低廉，谁将得到更多的用户。就第二条和第三条来看，现状还是如此，未来是否会出现低端的智能机，并达到中等功能应用的 Feature 手机价格的局面，值得大家期待。

2010 年 8 月 3 日，安卓在北美市场的份额中击败了苹果 iPhone，并且在 2010 年 4 月到 10 月这 6 个月期间，安卓成为美国用户智能手机的首选系统。美国移动广告网络公司 Millennial Media 公布的报告称，谷歌 Android 操作系统在全球智能手机的广告印象市场上占据主导地位，所占份额为 53%。在 56 个国家所做的市场调查中，Android 系统在 35 个国家市场占有率第一，平均市场占有率达到 48%，统领了整个亚太市场。

第 2 章　选择倚天剑还是屠龙刀

“武林至尊，宝刀屠龙，号令天下，莫敢不从，倚天不出，谁与争锋！”这是出自金庸先生的武侠名著《倚天屠龙记》中的一句话。屠龙刀和倚天剑锋利无比，传说谁得到谁就能够称霸江湖，一时间引起了江湖的血雨腥风。说到底，屠龙刀和倚天剑只是一种武器而已，即一种工具。现在无论做什么，都需要一种工具。行走江湖需要刀剑来防身，商人需要用账本来记账，农民需要用农具种庄稼……。同样 Android 开发也需要选择合适的工具。本章将详细介绍配置 Android 开发工具的具体过程，让读者根据需求来选择自己的屠龙刀或倚天剑。

2.1　工欲善其事，必先利其器

在正式拜师学艺后，师傅送了我一件绝世开发武器——Android SDK，师傅对我的期望很高，希望我好好修炼。“工欲善其事，必先利其器”意思是说要高效地完成一件事情，就要有一件合适的工具。对于我们安卓派的开发人员来说，开发武器同样至关重要，作为一项新兴技术，在进行开发前，首先要搭建一个开发环境。而在搭建开发环境前，需要先了解安装开发工具所需要的硬件和软件配置。

2.1.1　安装 Android SDK 的系统要求

在搭建开发环境之前，一定要先确定基于 Android 应用软件所需要的开发环境，具体如表 2-1 所示。

表 2-1　开发环境参数

项　目	版本要求	说　明	备 注
操作系统	Windows XP 或 Vista Mac OS X 10.4.8+Linux Ubuntu Drapper	根据自己的电脑自行选择操作系统	选择自己最熟悉的操作系统
软件开发包	Android SDK	选择最新版本的 SDK	截止到目前，SDK 最新版是 4.0

续表

项　目	版本要求	说　明	备 注
IDE	Eclipse IDE+ADT	Eclipse 3.3 (Europa)，3.4 (Ganymede) ADT(Android Development Tools)开发插件	选择“for Java Developer”版本
其他	JDK Apache Ant	Java SE Development Kit 5 或 6 Linux 和 Mac 上使用 Apache Ant 1.6.5+，Windows 上使用 1.7+版本	不能单独使用 JRE，开发 Android 必须有 JDK 的支持

因为当前主流的操作系统是 Windows，建议读者用 Eclipse+ADT 来作为开发工具。

2.1.2　Android 软件开发包

Android 软件开发包主要包括以下工具。

- JDK：可以到 http://java.sun.com/javase/downloads/index.jsp 处下载。
- Eclipse(Europa)：可以到 http://www.eclipse.org/downloads/处下载 Eclipse IDE for Java Developers。
- Android SDK：可以到 http://developer.android.com 处下载。
- 对应的开发插件。

2.2　用常规方法装备自己

2.2.1　依次安装 JDK、Eclipse、Android SDK

本书所讲的安装是以 Windows XP SP2 为平台，安装的软件为 JDK 1.6、Eclipse 3.5、ADT1.5、Android SDK 3.1。下面具体介绍各自的安装步骤(在配套光盘的视频中有详细的介绍)。

1．安装 JDK

第 1 步：安装 Eclipse 的开发环境需要 JRE 的支持，在 Windows 上安装 JRE/JDK 非常简单，首先在 Sun 官方网站下载，网址为 http://developers.sun.com/downloads/，如图 2-1 所示。

第 2 步：在图 2-1 中可以看到有很多版本，运行 Eclipse 时虽然只需要 JRE 就可以了，但是在开发 Android 应用程序时，需要完整的 JDK(JDK 已经包含了 JRE)，且要求其版本在 1.5+以上，这里选择 Java SE (JDK) 6，其下载页面如图 2-2 所示。

第 3 步：在图 2-2 中找到“JDK 6 Update 22”，单击其右侧的 Download JDK 按钮，出现让用户选择其操作系统和语言的界面，在此首先选择 Windows，然后单击 Download 按钮，如图 2-3 所示。

经过上述操作后，就已经开始下载安装文件 jdk-6u22-windows-i586.exe。

第 4 步：下载完成后，双击 jdk-6u22-windows-i586.exe 文件开始进行安装，首先弹出“安装向导”对话框，在此单击“下一步”按钮，如图 2-4 所示。

第 5 步：弹出“安装路径”对话框，在此选择文件的安装路径，选择“自定义安装”选项，

如图 2-5 所示。

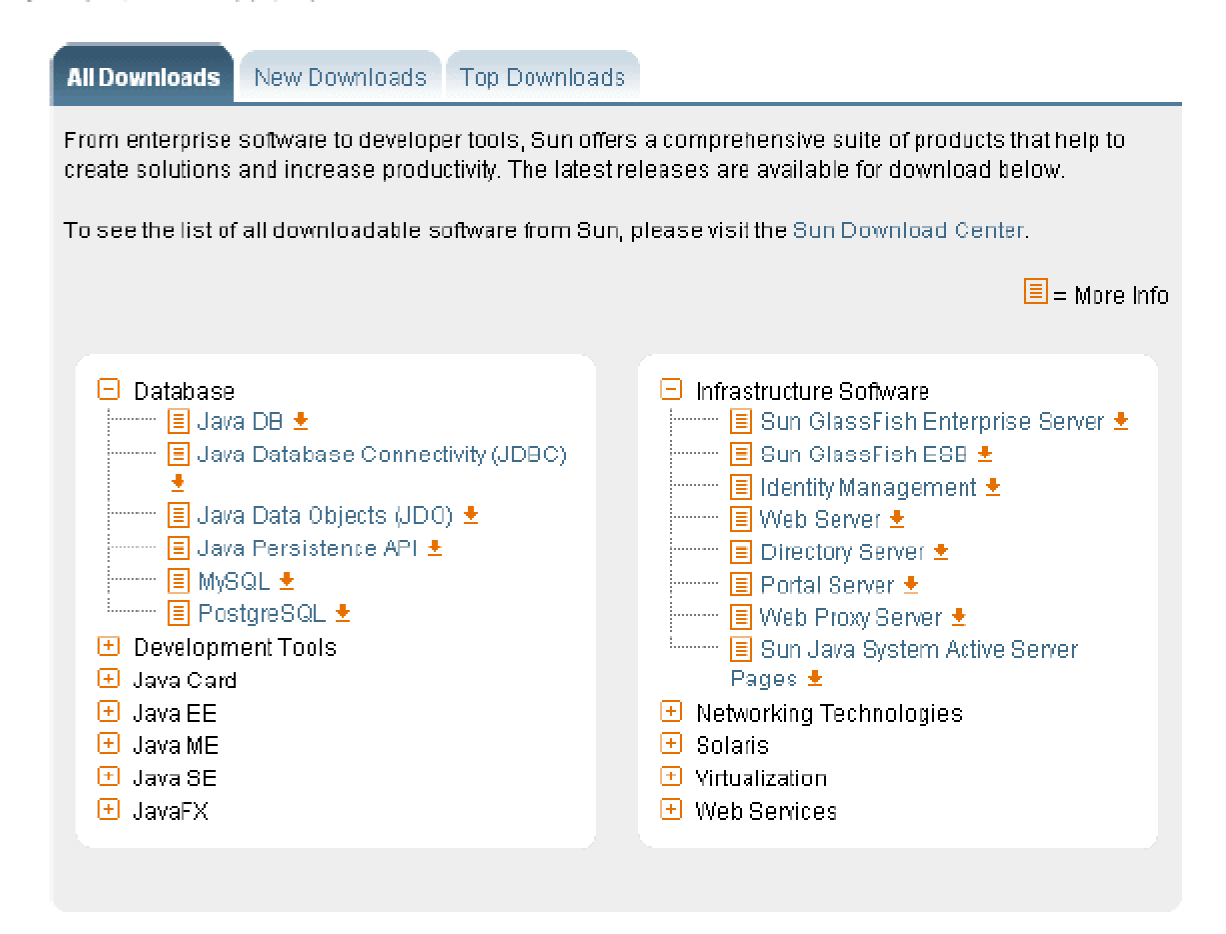


图 2-1　Sun 官方下载页面

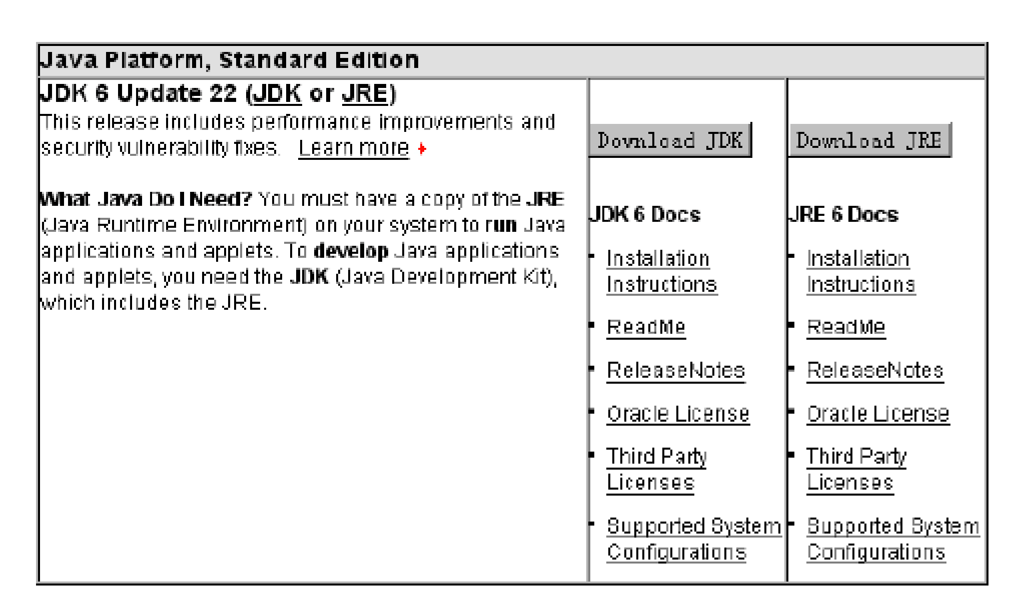


图 2-2　JDK 下载页面

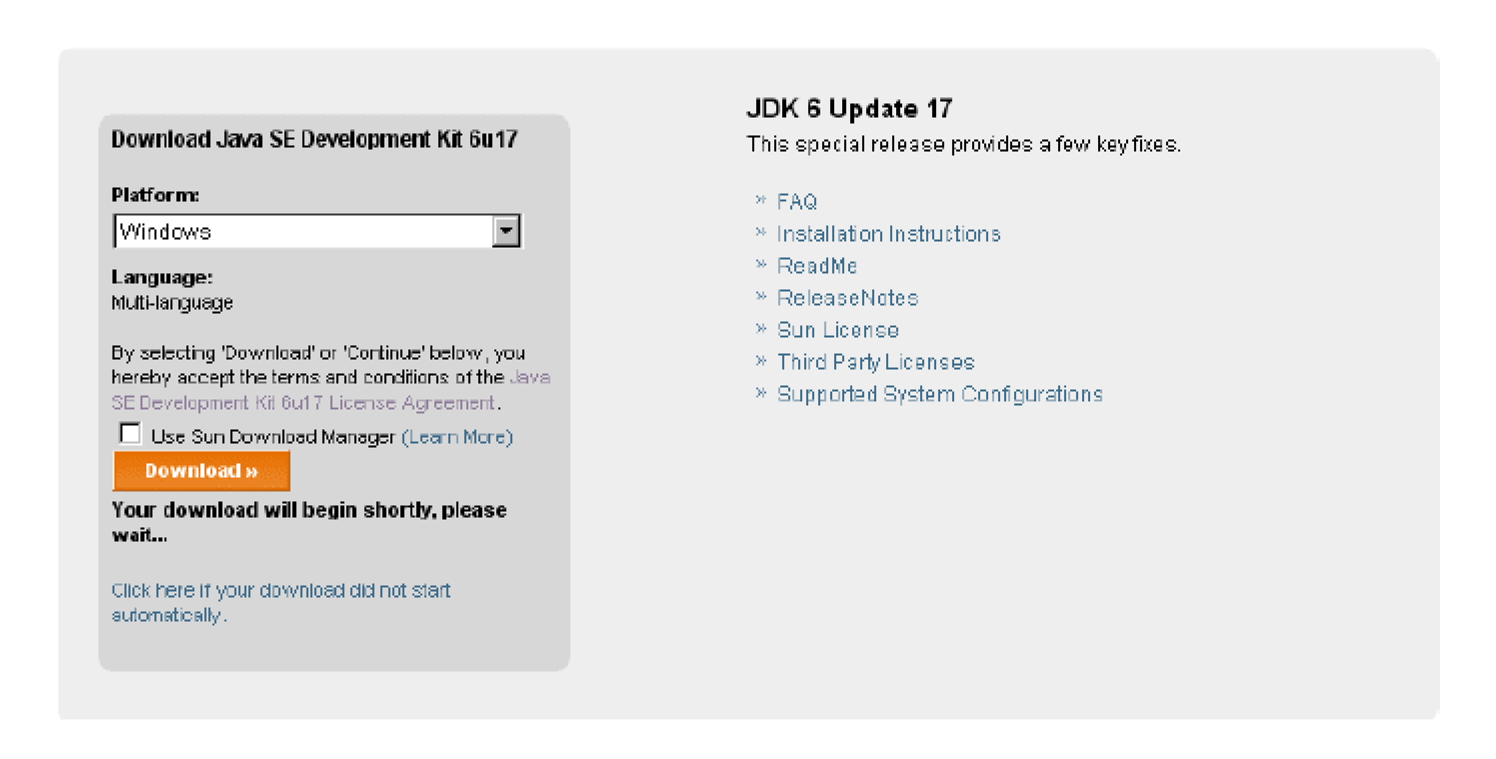


图 2-3　选择“Windows”



图 2-4　“安装向导”对话框



图 2-5　“自定义安装”对话框

第 6 步：单击“下一步”按钮，开始进行安装，如图 2-6 所示。

第 7 步：完成后弹出“目标文件夹”对话框，在此选择目标文件夹路径，如图 2-7 所示。

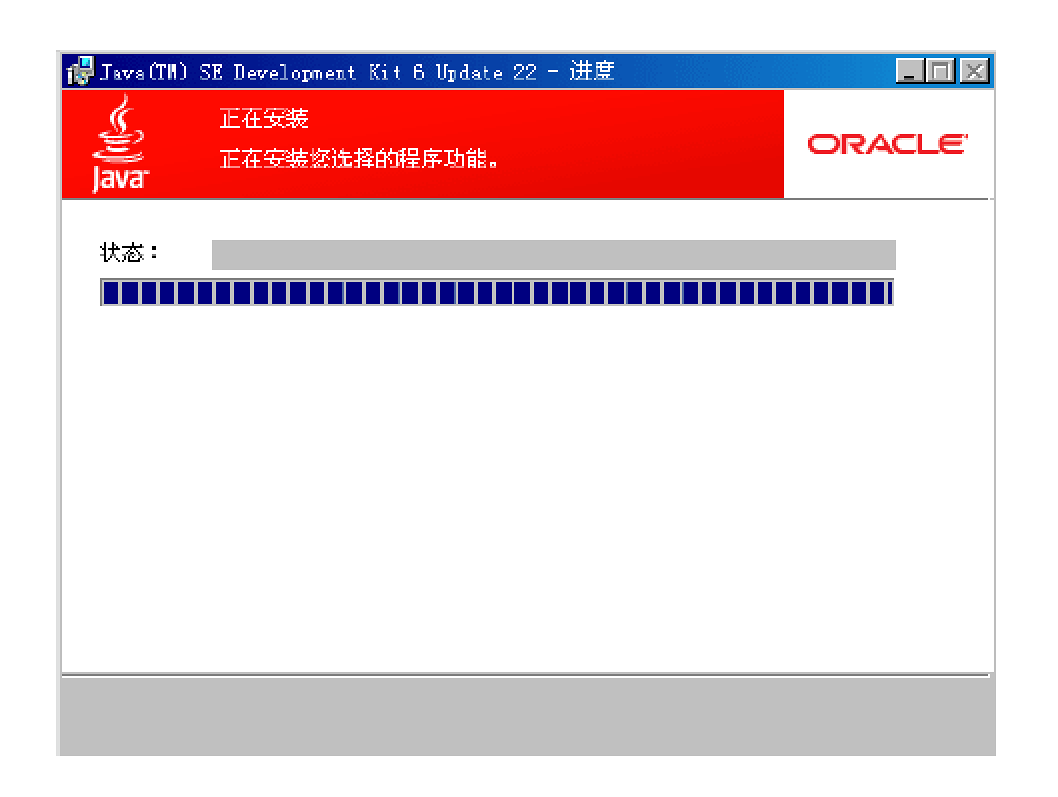


图 2-6　安装进度



图 2-7　“目标文件夹”对话框

第 8 步：单击“下一步”按钮后继续安装，如图 2-8 所示。

第 9 步：弹出“完成”对话框，单击“完成”按钮，完成整个安装过程，如图 2-9 所示。



图 2-8　继续安装



图 2-9　完成安装

完成安装后，我们可以检测是否安装成功，具体的方法是：依次选择“开始”|“运行”菜单命令，打开“运行”对话框，在对话框中输入“cmd”并按 Enter 键，在打开的 CMD 窗口中输入“java –version”，如果显示如图 2-10 所示的提示信息，则说明安装成功。

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>java -version
java version "1.6.0_22"
Java(TM) SE Runtime Environment (build 1.6.0_22-b04)
Java HotSpot(TM) Client VM (build 17.1-b03, mixed mode, sharing)

C:\Documents and Settings\Administrator>
```

图 2-10　CMD 窗口

如果检测到没有安装成功，则需要将其目录的绝对路径添加到系统的 PATH 中，具体添加方法如下。

第 1 步：右击“我的电脑”图标，选择“属性”命令，在弹出的“系统属性”对话框中，单击“高级”标签，切换到“高级”选项卡，单击下面的“环境变量”按钮，在“环境变量”对话框的“系统变量”选项组中单击“新建”按钮，在“编辑系统变量”对话框的“变量名”文本框中输入“JAVA_HOME”，在“变量值”文本框中输入刚才的目录，比如“C:\Program

Files\Java\jdk1.6.0_23”，如图 2-11 所示。

第 2 步：再新建一个变量名为 classpath，变量值为“.;%JAVA_HOME%/lib/rt.jar;%JAVA_HOME%/lib/tools.jar”的系统变量，确定后找到 PATH 的变量，双击或单击该变量进行编辑，在变量值最前面加上“%JAVA_HOME%/bin;”，如图 2-12 所示。



图 2-11　设置系统变量

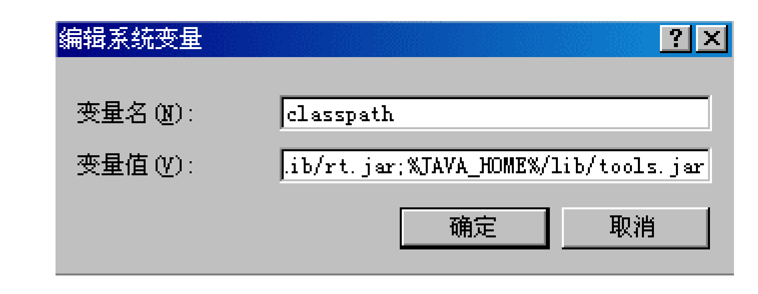


图 2-12　设置系统变量

第 3 步：再次依次选择“开始”|“运行”菜单命令，在运行框中输入“cmd”并按下 Enter 键，在打开的 CMD 窗口中输入“java –version”，如果显示如图 2-13 所示的提示信息，则说明安装成功。

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>java -version
java version "1.6.0_23"
Java(TM) SE Runtime Environment (build 1.6.0_23-b05)
Java HotSpot(TM) Client VM (build 19.0-b09, mixed mode, sharing)

C:\Documents and Settings\Administrator>_
```

图 2-13　CMD 窗口

注意： 上述变量设置是按照笔者本人的安装路径进行设置的，笔者安装 JDK 的路径是 C:\Program Files\Java\jdk1.6.0_22。当读者学习本书内容时，JDK 应该有更新的版本了。建议读者选择安装 JDK 的最新版本，安装和设置方法与上述过程完全一致。

2．安装 Eclipse

在安装好 JDK 后，就可以接着安装 Eclipse 了，具体安装过程如下。

第 1 步：打开 Eclipse 的下载页面 http://www.eclipse.org/downloads/，如图 2-14 所示。

第 2 步：在图 2-14 中选择“Eclipse IDE for Java Developers (92 MB)”，在其下载镜像页面选择离用户最近的镜像即可(一般推荐的下载速度就不错)，如图 2-15 所示。

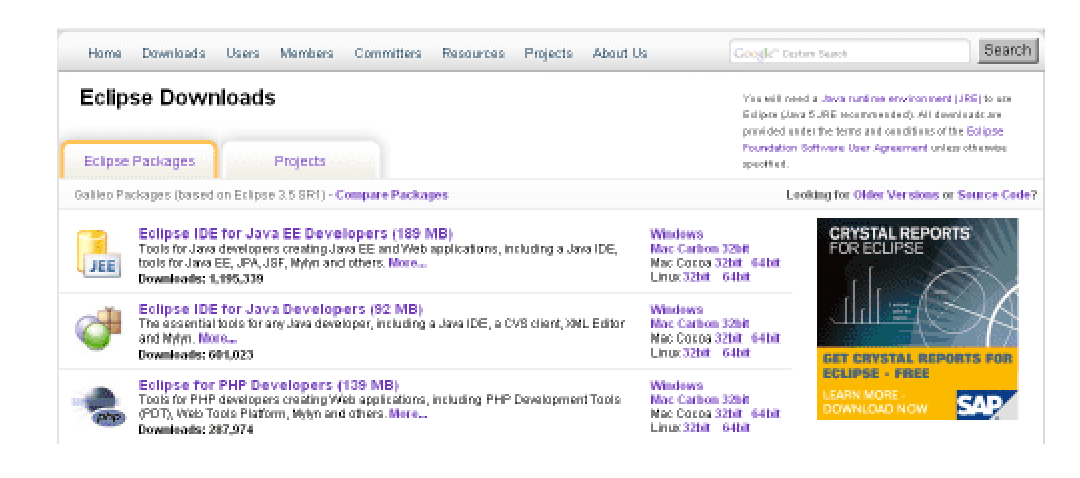


图 2-14　下载页面

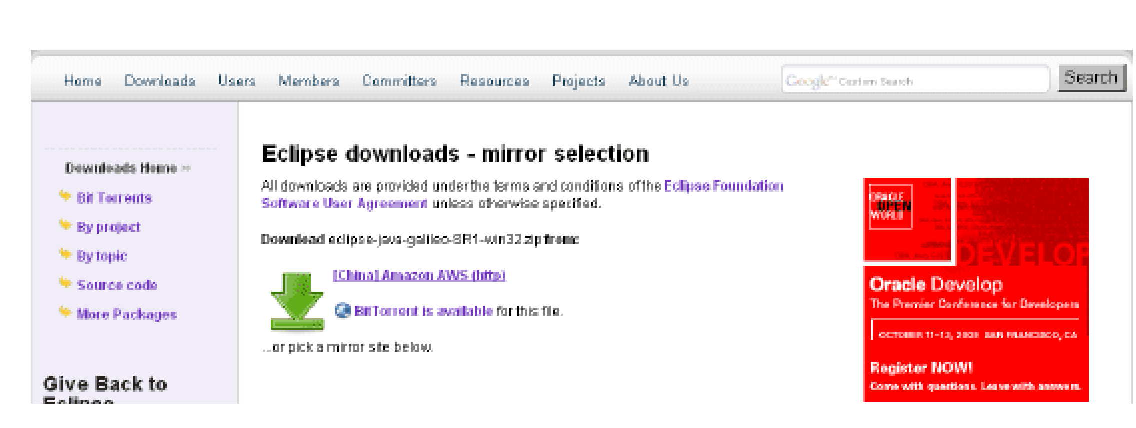


图 2-15　选择镜像

第 3 步：下载完成后，找到下载的压缩包“eclipse-java-galileo-SR1-win32.zip”。Eclipse 无须执行安装程序，直接解压此压缩文件就可以用，不过一定要先安装 JDK。在此假设 Eclipse 解压后存放的目录为 F:\eclipse。

第 4 步：进入解压后的目录，此时可以看到一个名为“eclipse.exe”的可执行文件，双击此文件直接运行，Eclipse 能自动找到用户先期安装的 JDK 路径，启动界面如图 2-16 所示。

第 5 步：因为是第一次安装、启动 Eclipse，会看到选择工作空间的提示，如图 2-17 所示。此时，单击 OK 按钮后即可完成 Eclipse 的安装。

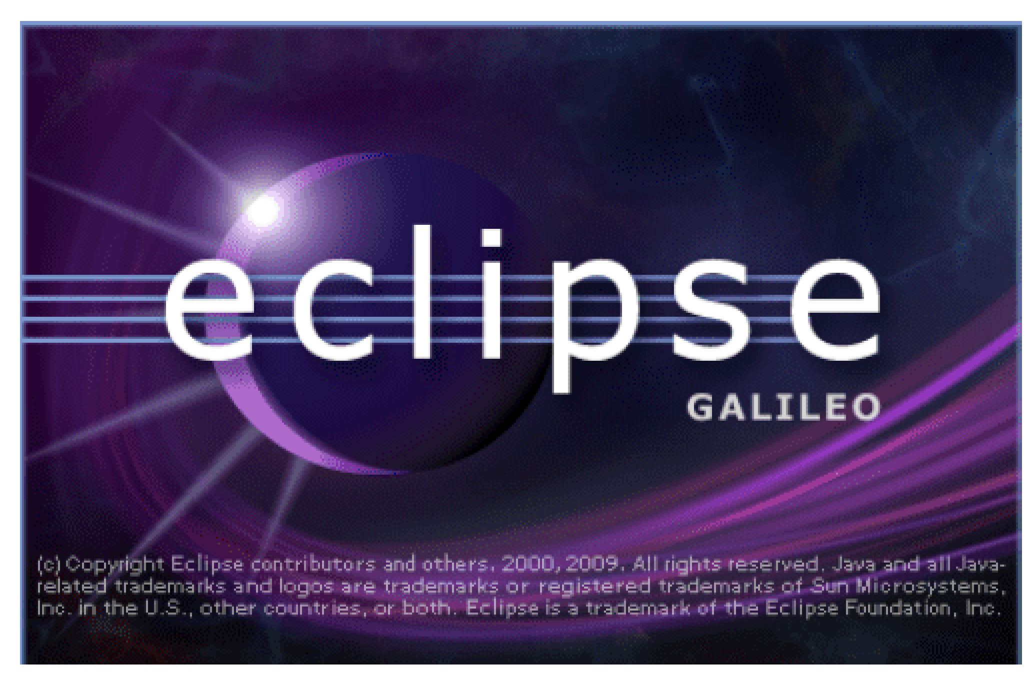


图 2-16　Eclipse 启动界面

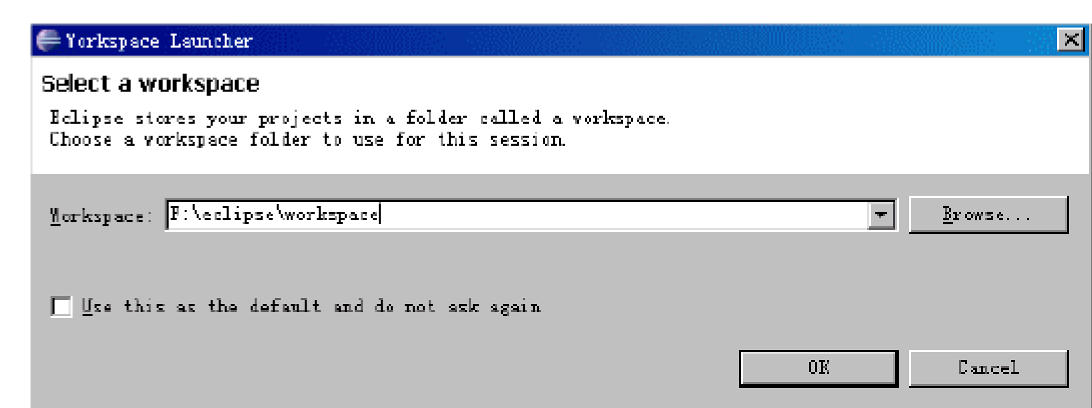


图 2-17　选择工作空间

3. 安装 Android SDK

安装完 JDK 和 Eclipse 后，接下来需要下载安装 Android 的 SDK，具体流程如下。

第 1 步：打开 Android 开发者社区网址 http://developer.android.com/，然后转到 SDK 下载页面，下载页面中显示的是最新的 Android SDK 3.1 的下载链接，如图 2-18 所示。

Download the Android SDK

Welcome Developers! If you are new to the Android SDK, please read the steps below, for an overview of how to set up the SDK.

If you're already using the Android SDK, you should update to the latest tools or platform using the *Android SDK and AVD Manager*, a new SDK starter package. See Adding SDK Components.

Platform	Package	Size	MD5 Checksum
Windows	android-sdk_r11-windows.zip	32837554 bytes	0a2c52b8f8d97a4871ce8b3eb38e3072
	installer_r11-windows.exe (Recommended)	32883649 bytes	3dc8a29ae5afed97b40910ef153caa2b
Mac OS X (intel)	android-sdk_r11-mac_x86.zip	28844968 bytes	85bed5ed25aea51f6a447a674d637d1e
Linux (i386)	android-sdk_r11-linux_x86.tgz	26984929 bytes	026c67f82627a3a70efb197ca3360d0a

Here's an overview of the steps you must follow to set up the Android SDK:

1. Prepare your development computer and ensure it meets the system requirements.
2. Install the SDK starter package from the table above. (If you're on Windows, download the installer for help with the initial setup.)

图 2-18　Android SDK 3.1 下载页面

第 2 步：在此选择用于 Windows 平台的“android-sdk_r11-windows.zip”，打开下载页面，如图 2-19 所示。

此时选中“I agree to the terms of the Android SDK License Agreement”复选框，然后单击 Download 按钮开始下载。

第 3 步：下载完成后，解压压缩文件。假设下载后的文件解压存放在 F:\android\目录下，并将其 tools 目录的绝对路径添加到系统的 PATH 中，具体操作方法如下。

(1)　右击“我的电脑”图标，选择“属性”命令，在弹出的“系统属性”对话框中单击“高级”标签，单击下面的“环境变量”按钮，在下面的“系统变量”选项组中选择“新建”选项，在“变量名”文本框中输入“SDK_HOME”，在“变量值”文本框中输入路径，比如

“F:\android-sdk-windows”，如图 2-20 所示。

(2) 找到 PATH 的变量，双击该变量后可以对它进行编辑，在变量值最前面加上“%SDK_HOME%\tools;”，如图 2-21 所示。

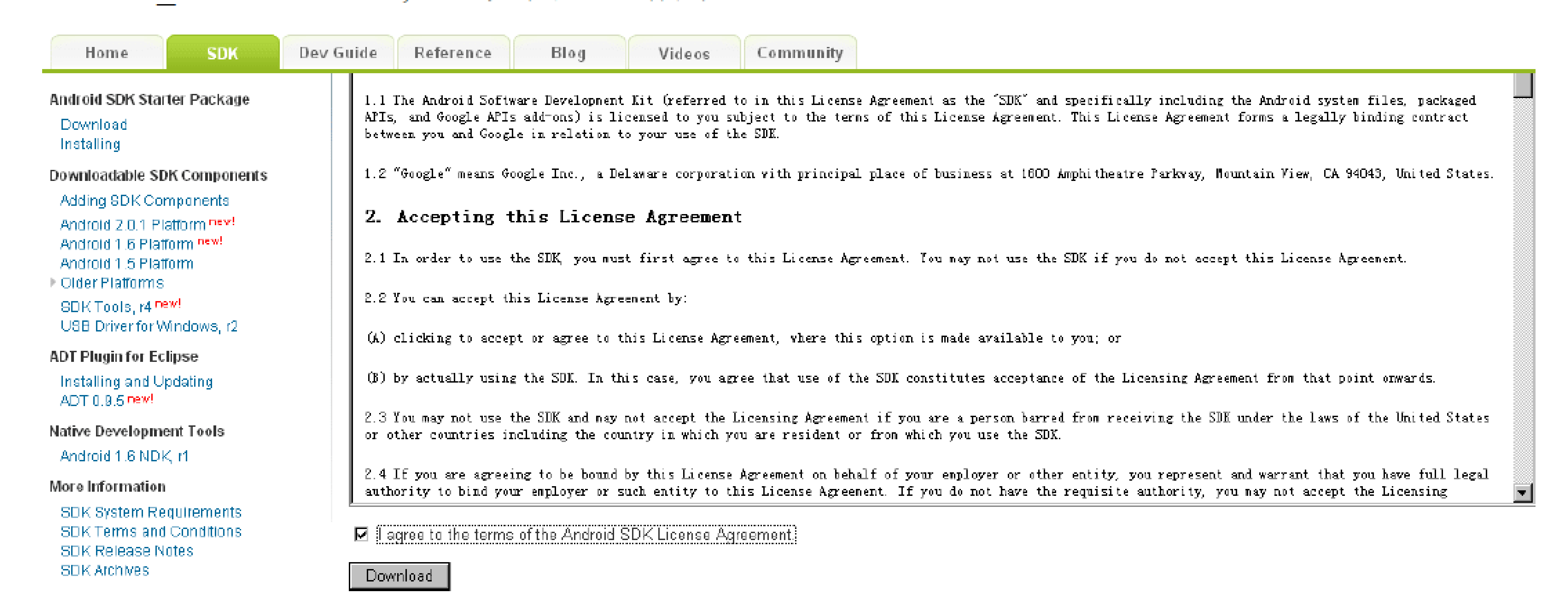


图 2-19　Android　SDK 下载页面



图 2-20　设置系统变量

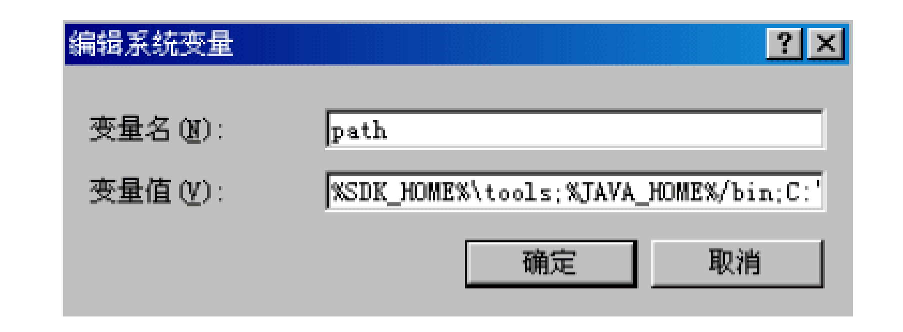


图 2-21　设置系统变量

(3) 依次选择“开始”|“运行”菜单命令，在“打开”文本框中输入“cmd”并按 Enter 键，在打开的 CMD 窗口中输入一条测试命令，例如 android -h，当显示如图 2-22 所示的提示信息，则说明安装成功。

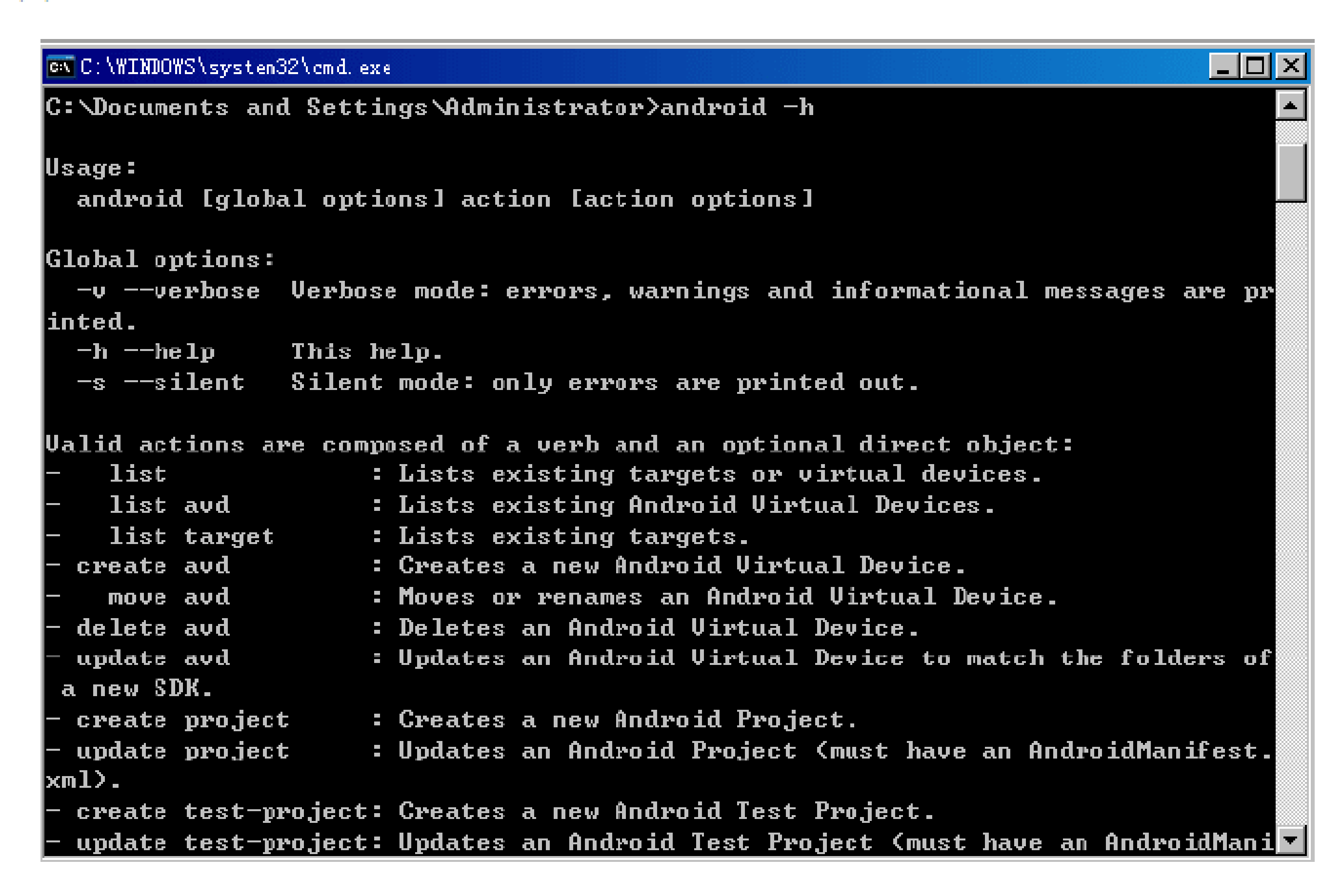


图 2-22　设置系统变量

4．安装 ADT

Android 为 Eclipse 定制了一个插件，即 Android Development Tools(ADT)，这个插件为用户提供了一个强大的综合环境，用于开发 Android 应用程序。ADT 扩展了 Eclipse 的功能，可以让

用户快速地建立 Android 项目，创建应用程序界面，在基于 Android 框架 API 的基础上添加组件，以及用 SDK 工具集调试应用程序，甚至导出签名(或未签名)的 APKs 以便发行应用程序。下面详细介绍安装配置 ADT 的基本方法。安装 Android Development Tools plug-in，的操作步骤如下。

第 1 步：打开 Eclipse 后，依次选择 Help | Install New Software 菜单命令，如图 2-23 所示。

第 2 步：在弹出的 Install 对话框中单击 Add 按钮，如图 2-24 所示。

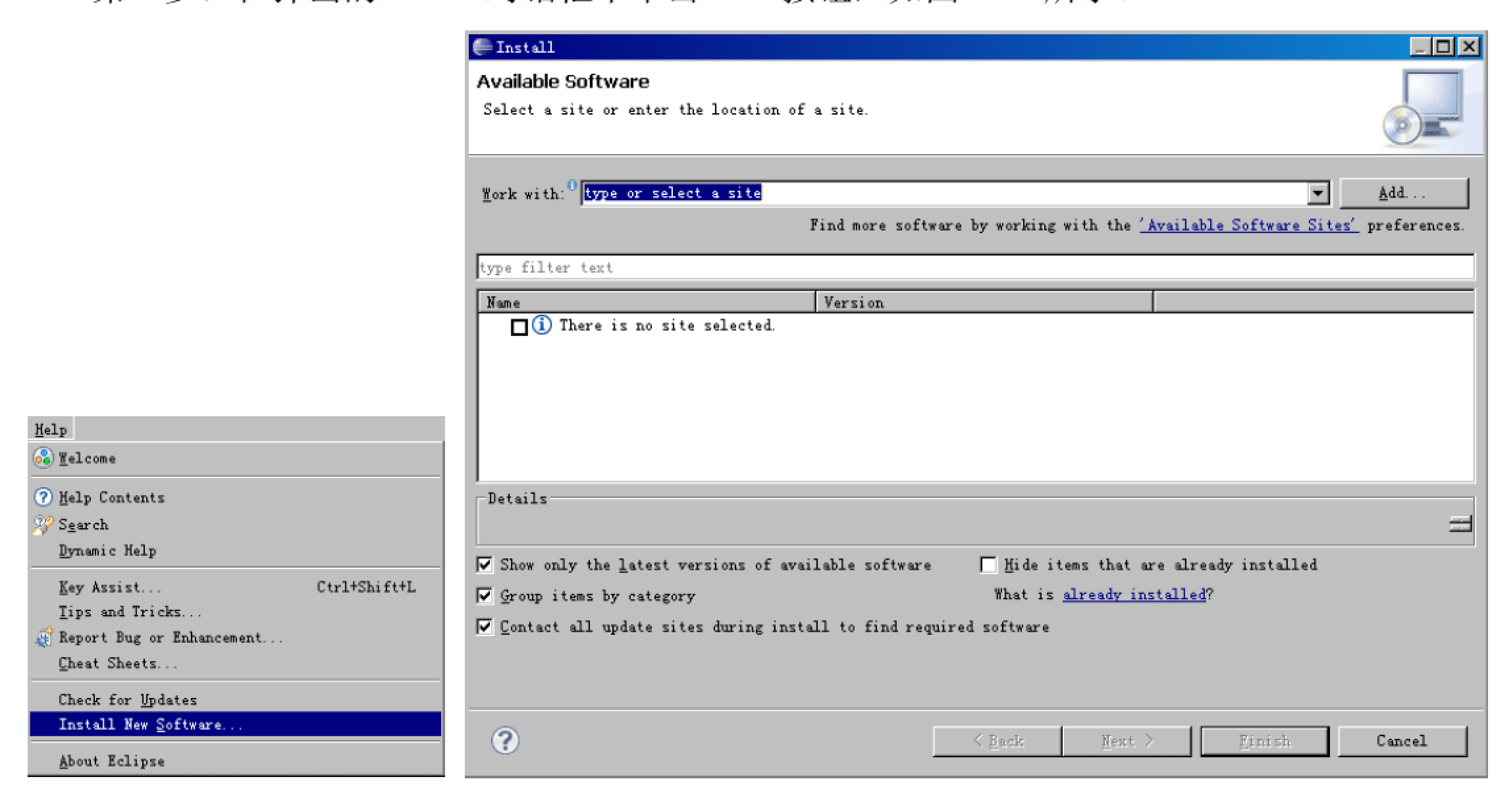


图 2-23　添加插件　　　　图 2-24　添加插件

第 3 步：在弹出的 Add Site 对话框中分别输入名字和地址，名字可以自己命名，例如“123”，但是在 Location 文本框中必须输入插件的网络地址“http://dl-ssl.google.com/Android/eclipse/”，单击 OK 按钮，如图 2-25 所示。

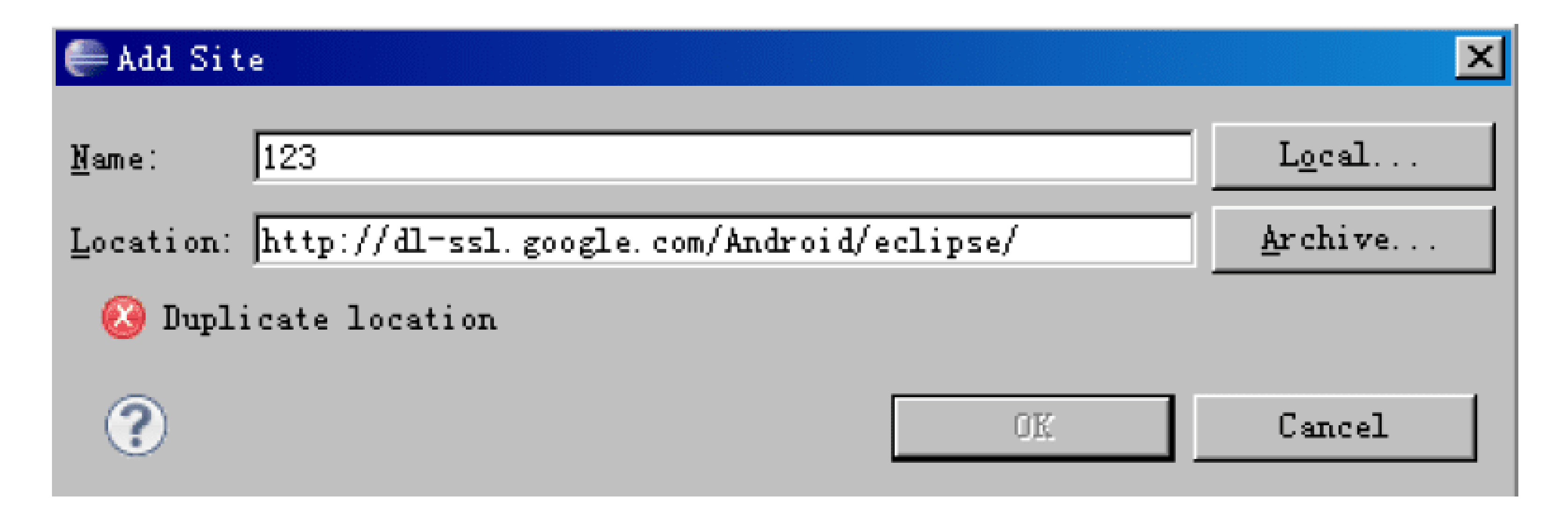


图 2-25　设置地址

第 4 步：单击 OK 按钮后完成设置，此时在 Install 对话框中将会显示可用的插件，如图 2-26 所示。

第 5 步：选择 Android DDMS 和 Android Development Tools 选项，然后单击 Next 按钮，进入插件安装界面，如图 2-27 所示。

第 6 步：选择“I accept the terms of the license agraments”单选按钮，然后单击 finish 按钮，则开始进行安装，如图 2-28 所示。

注意： 在以上操作步骤中，可能会发生计算插件占用资源的情况，过程有点慢。完成后，会提示重启 Eclipse 来加载插件，此时单击 OK 按钮重启就可以用了。不同版本的 Eclipse 安装插件的方法和步骤是不同的，但是大同小异，根据操作提示读者能够自行解决。

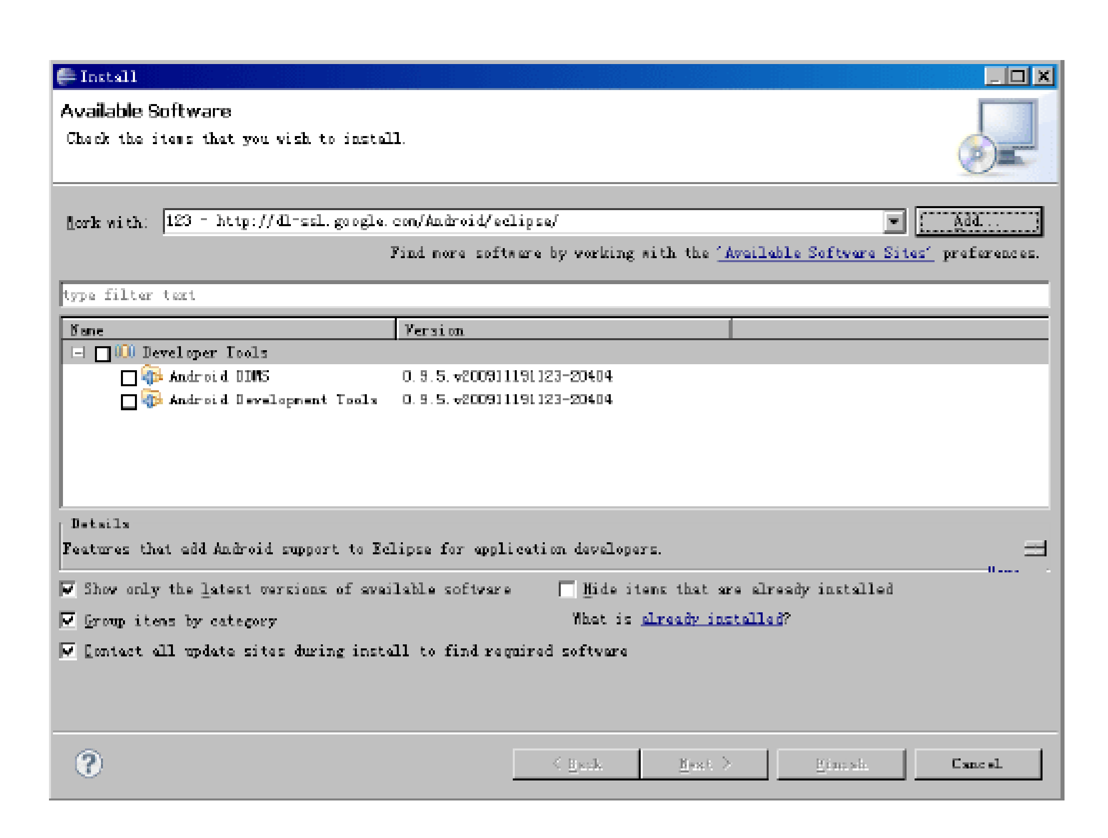


图 2-26　插件列表

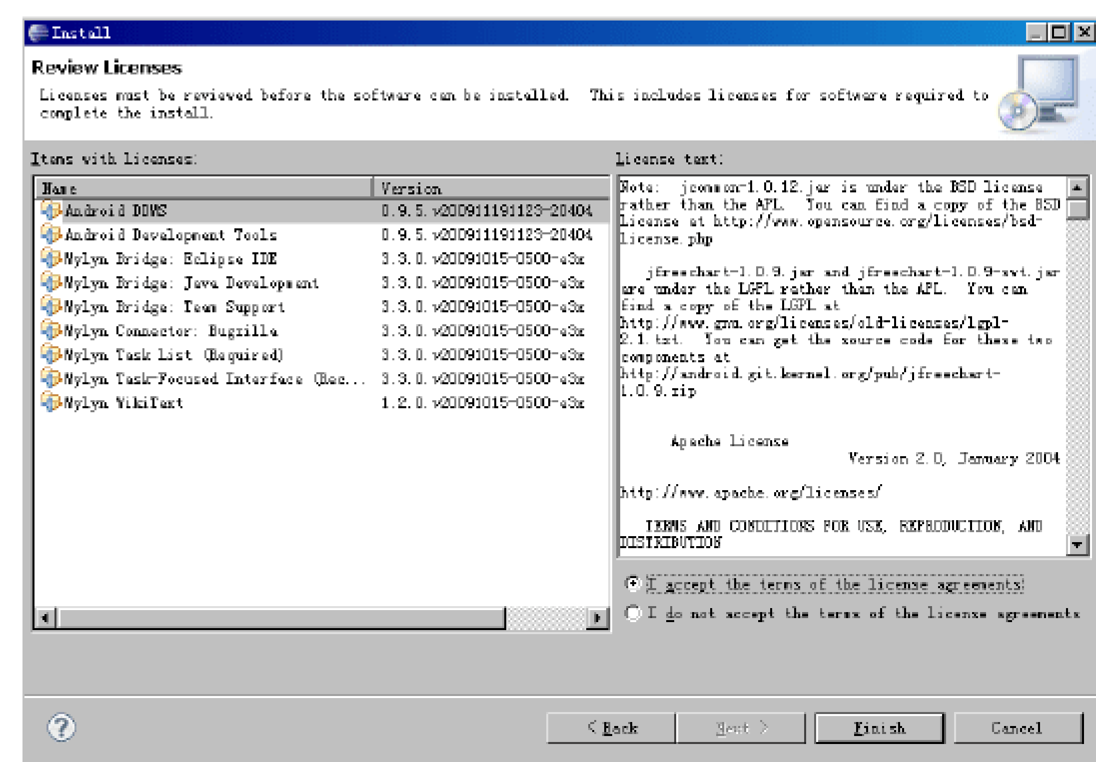


图 2-27　插件安装界面

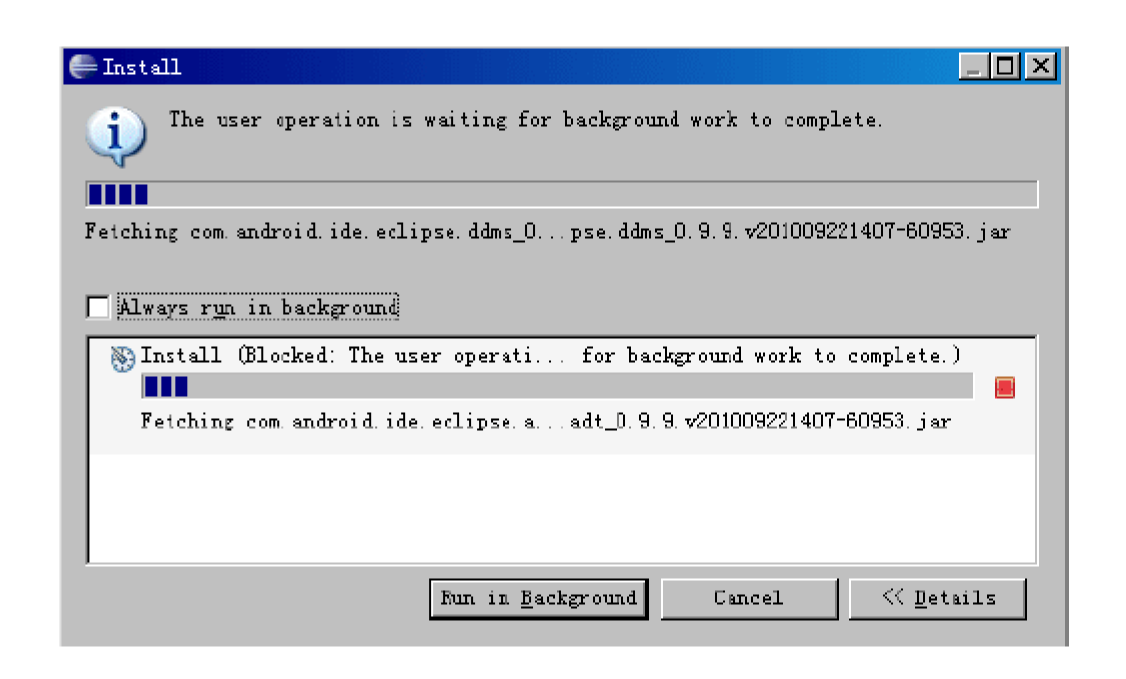


图 2-28　开始安装

2.2.2　设定 Android SDK Home

装备完插件后，我发现还不能使用 Eclipse 创建 Android 项目。无奈之下只好向师傅请教，师傅神秘地说：“你需要在 Eclipse 中设置 Android SDK 的主目录。”

设置 Android SDK 主目录的方法如下。

第 1 步：打开 Eclipse，在菜单中依次选择 Window | Preferences 菜单命令，如图 2-29 所示。

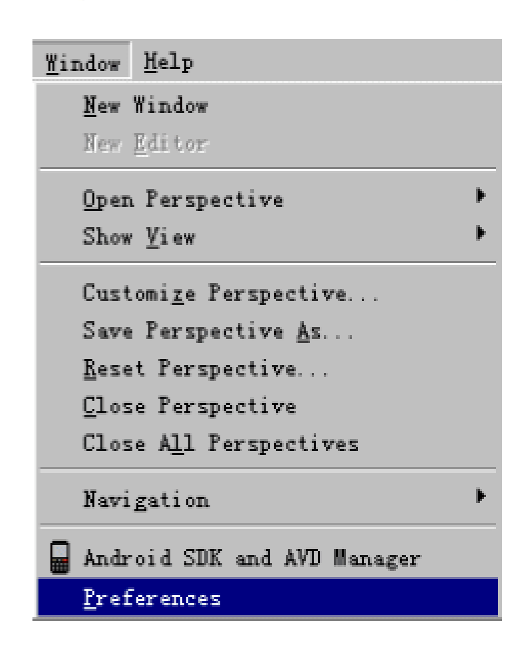


图 2-29　选择 Preferences 菜单命令

第 2 步：在弹出的界面左侧可以看到 Android 选项，选中 Android 选项后，在右侧设置 Android SDK 所在目录为 SDK Location，单击 OK 按钮完成安装，如图 2-30 所示。

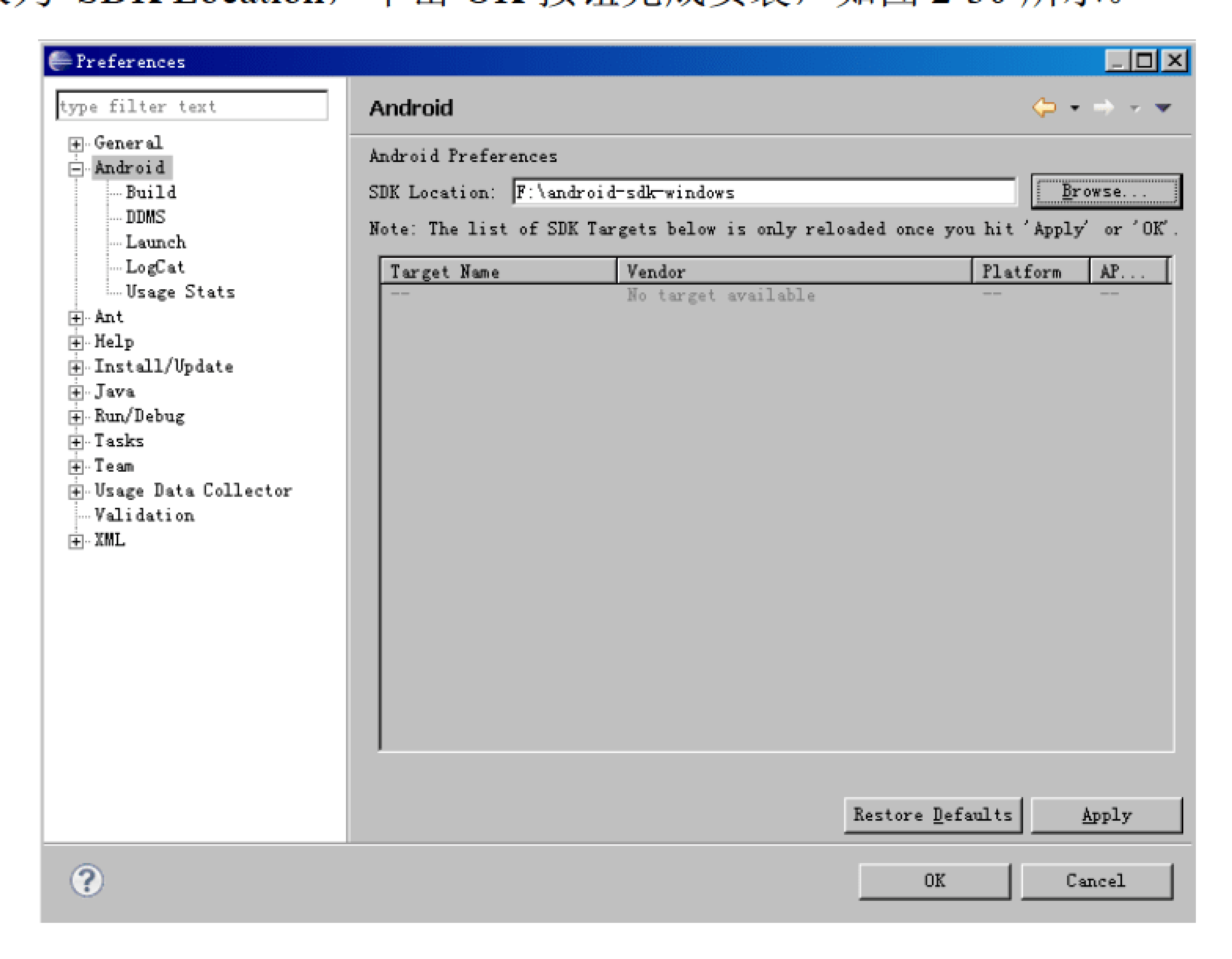


图 2-30 设置 Android SDK 所在目录

2.2.3 验证开发环境

经过本章前面知识的讲解，Android 开发环境已经搭建完成。下面通过新建一个项目来验证当前的环境是否可以正常工作，具体操作如下。

第 1 步：打开 Eclipse，在菜单中依次选择 File | New | Project 菜单命令，在弹出的对话框中可以看到 Android 选项，如图 2-31 所示。

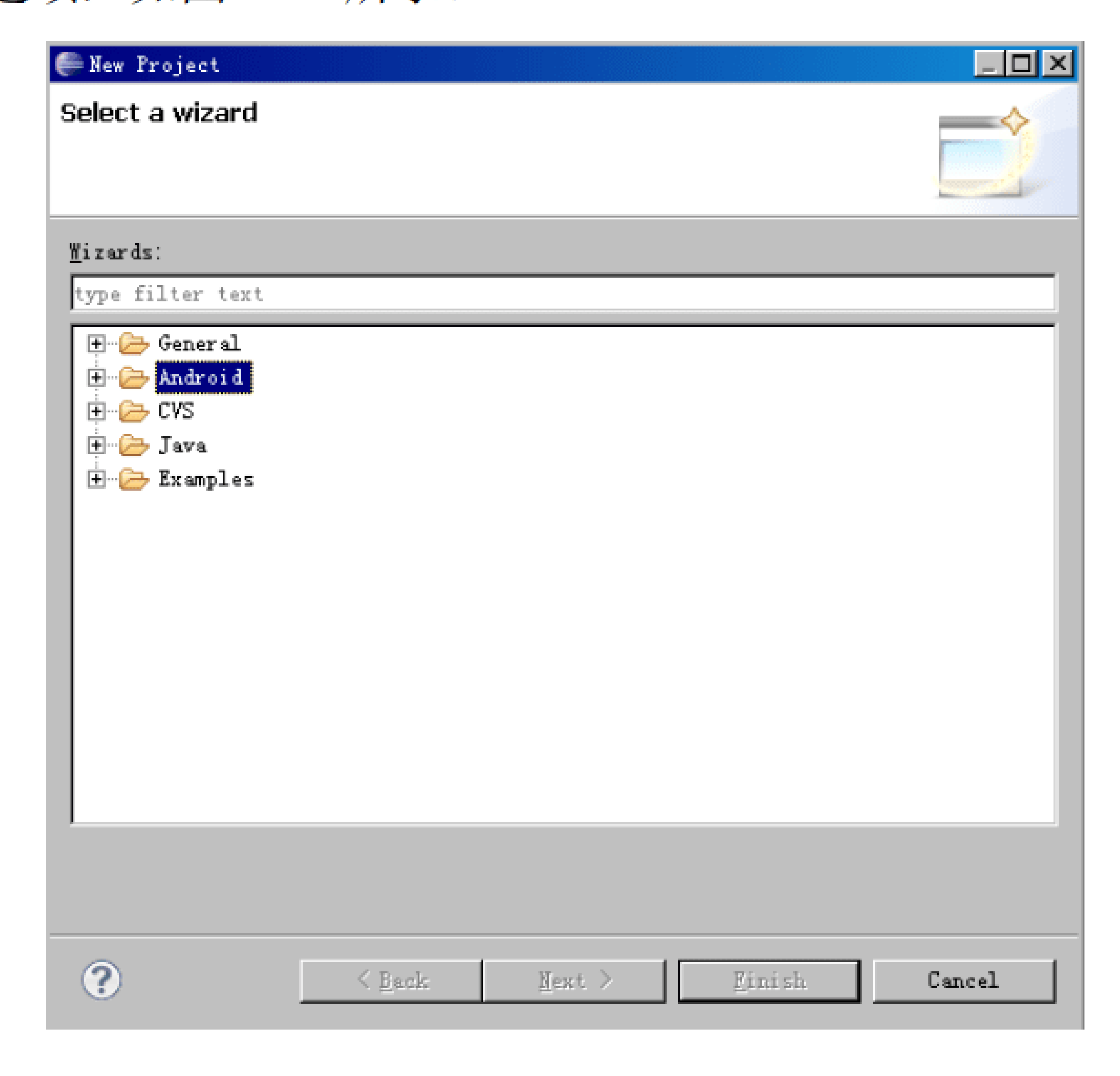


图 2-31 新建项目

第 2 步：在图 2-31 中选择 Android 选项，单击 Next 按钮，打开 New Android Project 对话框，在对应的文本框中输入必要的信息，如图 2-32 所示。

第 3 步：单击 Finish 按钮，会自动完成项目的创建工作，最后将可以看到如图 2-33 所示的项目结构。这里不再具体说明项目信息中各项的意义，后面章节中有详细介绍。

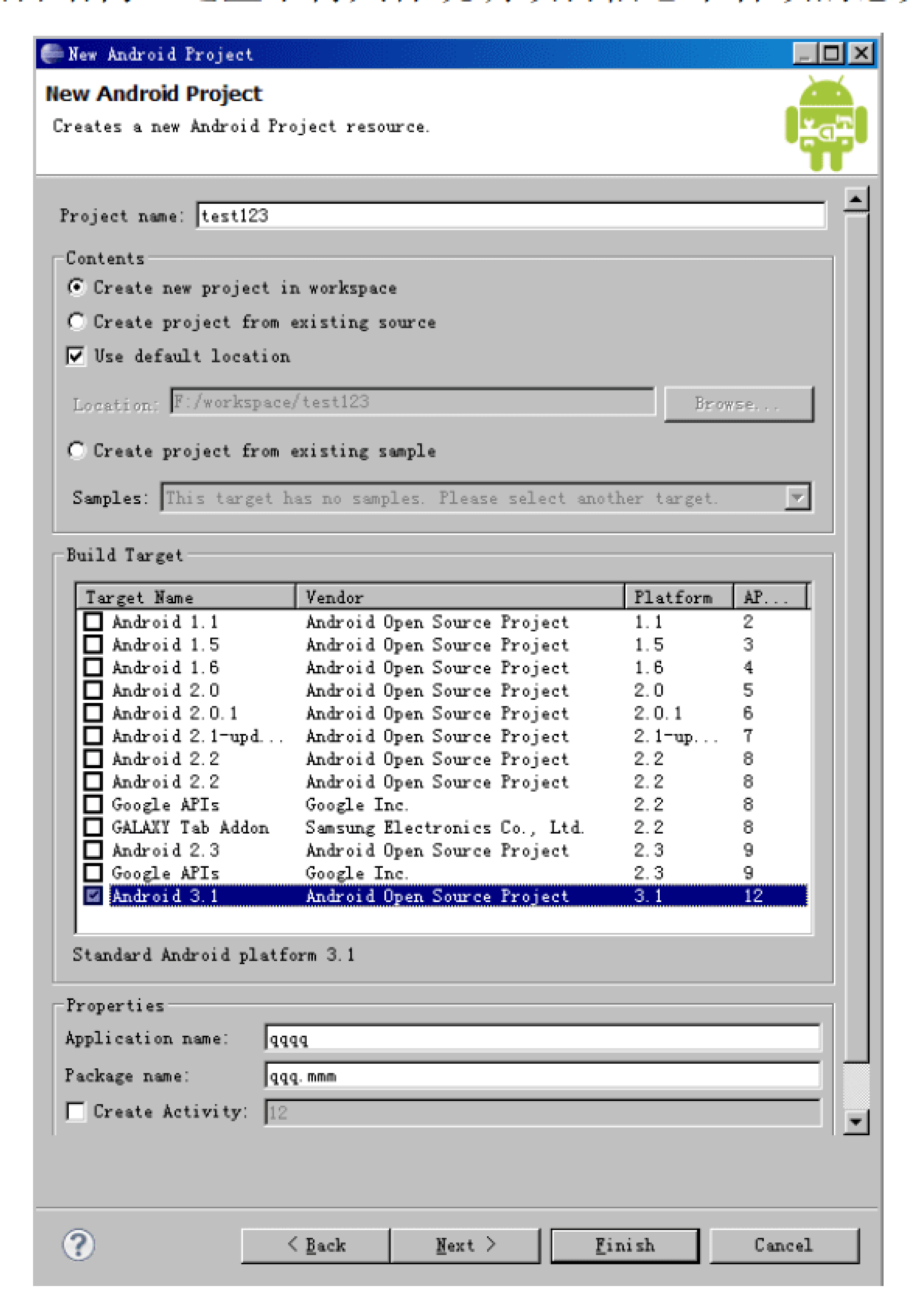


图 2-32　New Android Project 对话框

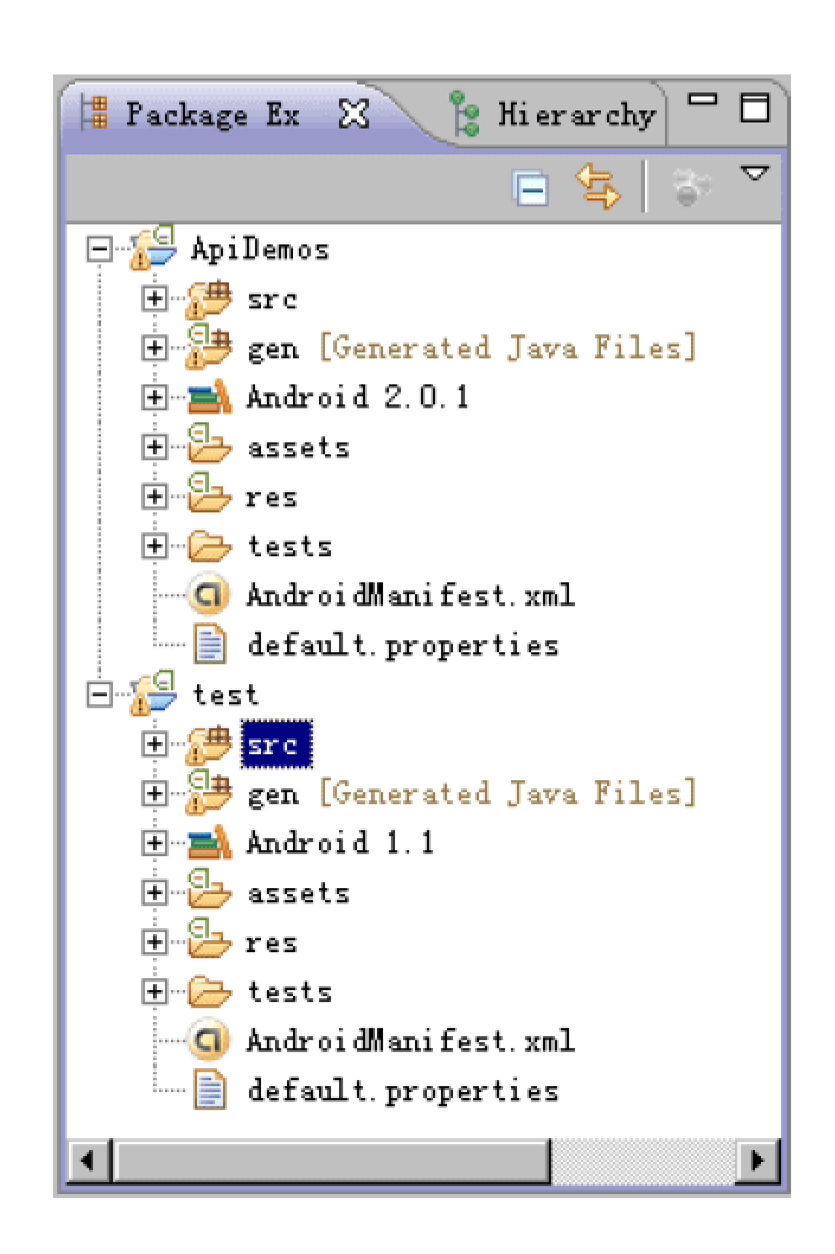


图 2-33　项目结构

2.2.4　创建 Android 虚拟设备(AVD)

AVD 全称为 Android 虚拟设备(Android Virtual Device)，每个 AVD 模拟了一套虚拟设备来运行 Android 平台，这个平台至少要有自己的内核、系统图像和数据分区，还可以有自己的 SD 卡和用户数据以及外观显示等。创建 AVD 的操作步骤如下。

第 1 步：在 CMD 下输入“android list targets”查看可用的平台，对应的 CMD 窗口如图 2-34 所示。图中列举了 13 个 targets，id 分别是 1～13。

第 2 步：创建 AVD。按照“android create avd --name <your_avd_name> --target <targetID>”格式创建 AVD，其中“your_avd_name”是需要创建的 AVD 的名字，对应的 CMD 窗口界面如图 2-35 所示。

第 3 步：这样就创建了一个自定义的 AVD，然后只要在 Eclipse 的 Run Configurations 中指定一个 AVD，即可在 Target 下选中自己定义的 AVD，这样 sdk_1_5_version 就可以运行了，如图 2-36 所示。

```
C:\WINDOWS\system32\cmd.exe
     Revision: 1
     Based on Android 2.2 (API level 8)
     Skins: WVGA854, WQVGA400, GALAXY Tab (default), HVGA, WQVGA432, WVGA800, QV
GA
id: 11 or "android-9"
     Name: Android 2.3
     Type: Platform
     API level: 9
     Revision: 1
     Skins: HVGA, QVGA, WQVGA400, WQVGA432, WVGA800 (default), WVGA854
id: 12 or "Google Inc.:Google APIs:9"
     Name: Google APIs
     Type: Add-On
     Vendor: Google Inc.
     Revision: 1
     Description: Android + Google APIs
     Based on Android 2.3 (API level 9)
     Libraries:
      * com.google.android.maps (maps.jar)
          API for Google Maps
     Skins: WVGA854, WQVGA400, HVGA, WQVGA432, WVGA800 (default), QVGA
id: 13 or "android-12"
     Name: Android 3.1
     Type: Platform
     API level: 12
     Revision: 1
     Skins: WXGA (default)
```

图 2-34　CMD 窗口

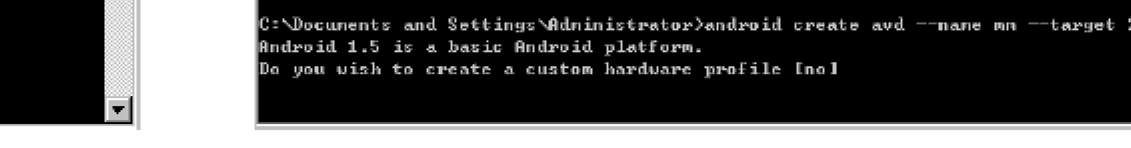


图 2-35　创建 AVD 时对应的 CMD 窗口

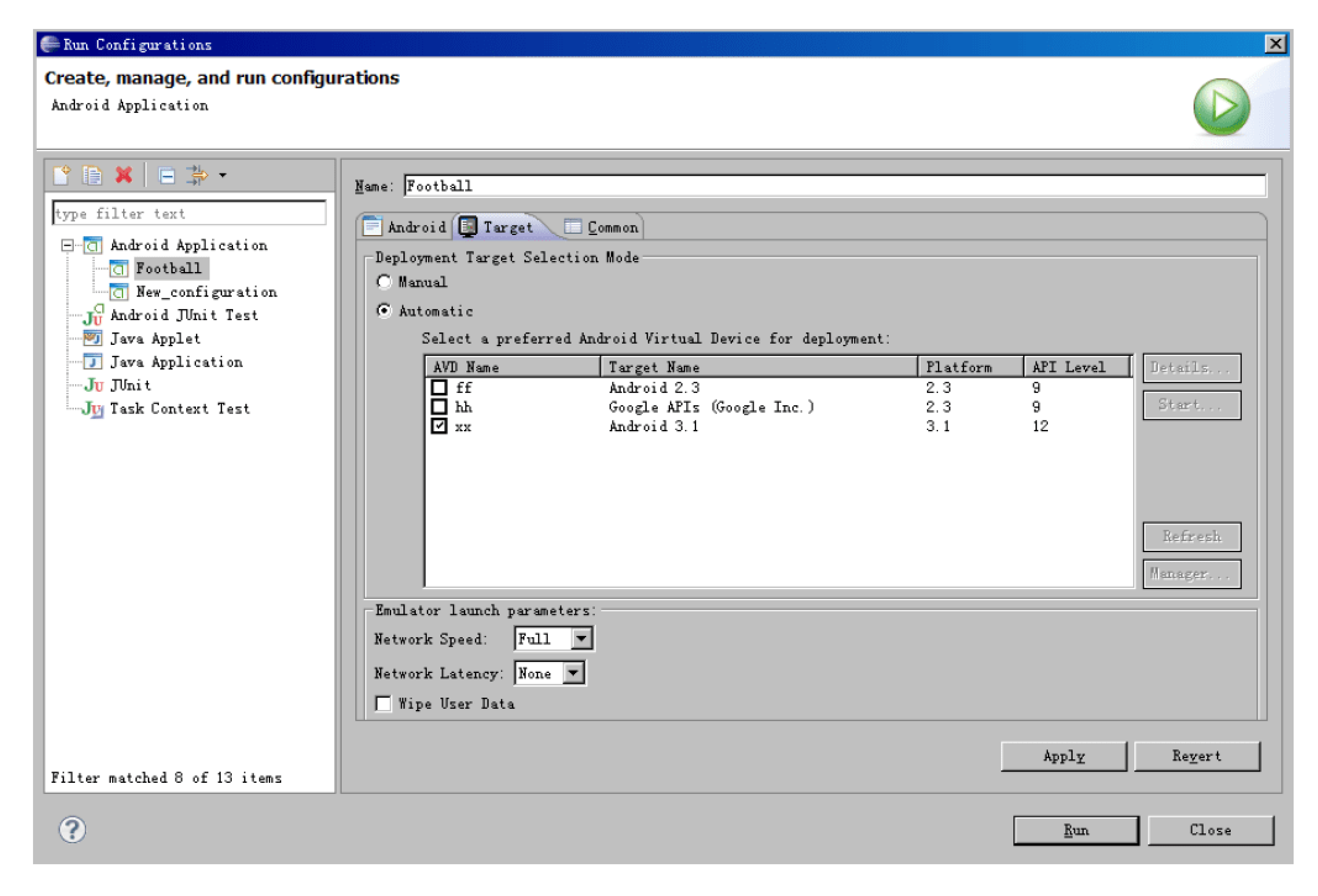


图 2-36　选择 AVD

第 4 步：单击 Apply 按钮，然后单击 Start 按钮弹出 Launch Options 对话框，如图 2-37 所示。

第 5 步：单击 Launch 按钮后将会运行模拟器，如图 2-38 所示。

关于应用结构分析和讲解以及代码的调试部分内容，会在本书后面的内容中进行详细介绍。至此，在 Windows 平台上的开发环境已经搭建完成，安装了运行环境 JDK、开发工具 Eclipse、Android SDK，安装了 ADT 并进行了 SDK Home 的配置，最后创建了一个 Android 虚拟设备(AVD)。

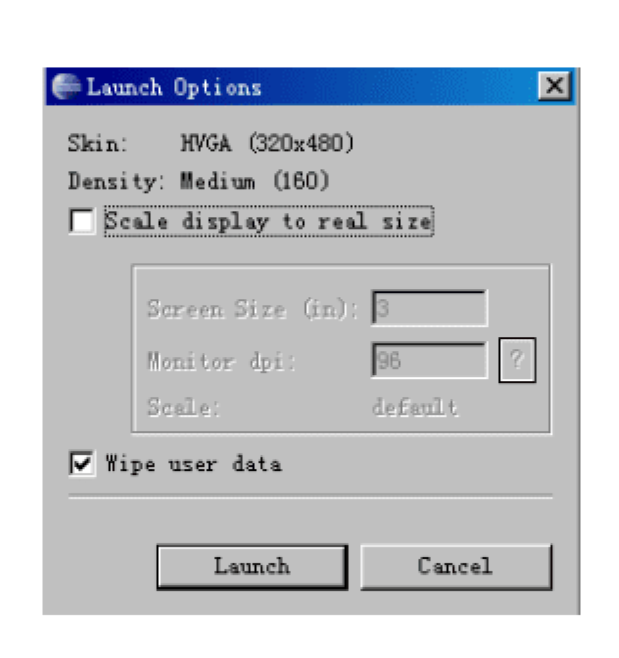


图 2-37　Launch Options 对话框

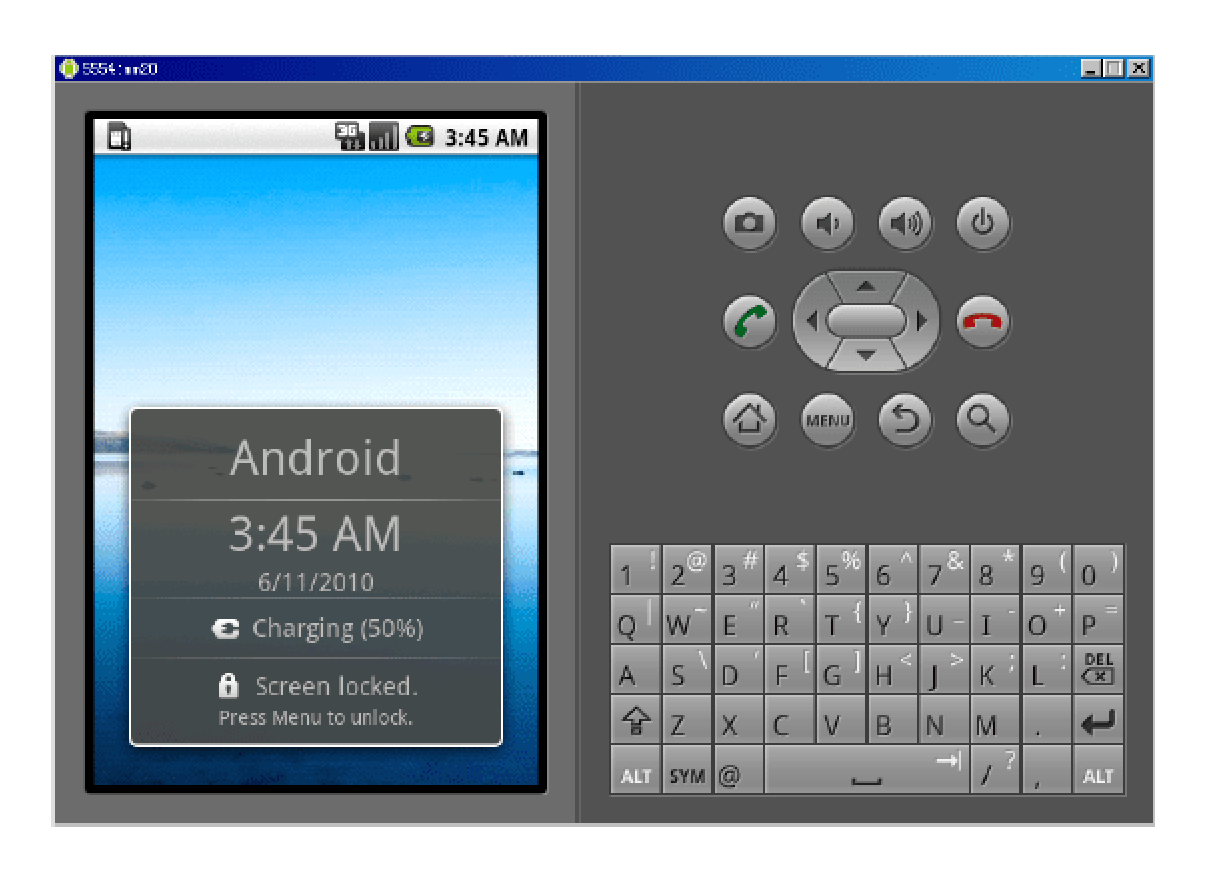


图 2-38　模拟器运行成功

2.3　不走寻常路

Android 知识体系博大精深，需要付出一些时间和精力才能真正掌握这门技术。有很多人为了急于求成，纷纷学习 Linux 环境下的开发知识。但是当前市场最迫切需求的是 Windows 下的 Android 人才，所以我决定专心学习 Windows 环境下的开发知识，至于 Linux 方面的相关开发知识，只需简单了解即可。

2.3.1　另辟蹊径之——Linux 下的搭建过程

下面以 Linux ubuntu 8.10 为平台，介绍搭建 Android 开发环境的具体方法，操作流程如下。

第 1 步：安装虚拟光驱 daemon400.exe。

第 2 步：在 Windows XP 下用虚拟光驱安装 ubuntu 8.10，iso 文件为：ubuntu-8.10-beta-desktop-i386.iso。

第 3 步：用 dpkg 命令打开 patch。首先进入 ubuntu 系统，将 ubuntu_package_0430.tar.gz 文件解压。

```
tar -zvxf ubuntu_package_0430.tar.gz
```

然后打开 patch：

```
sudo dpkg -i *.deb
```

如果存在没有成功的，则再依次执行：

```
sudo dpkg -i  filename.deb
```

注意：可能有时需要一起运行 dpkg，形式如下：

```
sudo dpkg -i  filename1.deb filename1.deb
```

另外，还需要重新用 Java 5 执行 dpkg 命令(因为用 Java 6 会有问题)。

第 4 步：编译原码和 Android sdk。

编译原码：解压原码到本地，进入原码目录。接着执行下面的代码：

```
make
```

编译 sdk：在 make 完成后，直接 make sdk，会在 out/host/linux-x86/sdk 下面生成 mdk 文件及文件夹，形如：android-sdk_eng.xxx_linux-x86。

第 5 步：安装 Eclipse。只需直接解压 eclipse-jee-ganymede-SR2-linux-gtk.tar.gz 即可：tar -zvxf eclipse-jee-ganymede-SR2-linux-gtk.tar.gz。

第 6 步：在 Eclipse 下配置 Android。可参考高焕堂编写的《Android 应用框架原理与程式设计 36 技》附录 A。

> **注意：** eclipse-jee-ganymede-SR2-linux-gtk.tar.gz 对应的 ADT 要用 ADT-0.9.1.zip，而不是 0.8.1 版本。

第 7 步：测试刚才编译好的 SDK。

(1) 在 Eclipse 中将 Android SDK 目录设置成自己编译生成的 SDK 目录，例如 out/host/linux-x86/sdk/android-sdk_eng.xxx_linux-x86。依次选择 Window | preferences | Android 菜单中的 SDK Location，并进行设置。

(2) 创建 AVD：在 Eclipse 中选择 Window | Android AVD Manager 菜单命令，将 name、target、sdcard、skin 都填选好后，单击 Create AVD 按钮即可。

(3) 在 cmd 窗口中进入到目录下，执行下面的命令：

```
emulator -avd avdname
```

经过上述操作后，模拟器就运行起来了。

> **注意：** 如果没有需要的 JDK、Eclipse 和 Android SDK，在 Linux 下也需要分别下载它们，只是在下载时选择 Linux 的资源即可，整个安装顺序和 Windows 下的大同小异。

2.3.2 另辟蹊径之——苹果下的搭建过程

在苹果系统下搭建 Android 的具体过程和 Linux 下的类似，为节省篇幅，将不再进行详细介绍。只是在下载 Android SDK 的时候，注意选择 Mac OS 版本的相关资源。

2.4 解决常见的安装问题

领到装备后，仅仅用了几天的工夫，我就对 Android SDK 熟悉得差不多了。在这个万物复苏的春天里，我刚要开始新一天的修炼，师傅就给我布置了一个作业：检讨自己在装备 Android SDK 时出现过的问题。俗话说严师出高徒，虽然我很无奈，但内心深处还是很乐意地接受了这个作业。

1. Android 不能在线更新

在安装 Android 后，需要更新为最新的资源和配置。但是在启动 Android 后，经常不能更新，

弹出如图 2-39 所示的错误提示。

Android 默认的在线更新地址是 https://dl-ssl.google.com/android/eclipse/，但是经常会出现错误。如果此地址不能更新，可以自行设置更新地址，修改为 http://dl-ssl.google.com/android/repository/repository.xml。具体操作方法如下。

(1)　单击 Android 左侧的 Available Packages 选项，然后单击 Add Site 按钮，如图 2-40 所示。

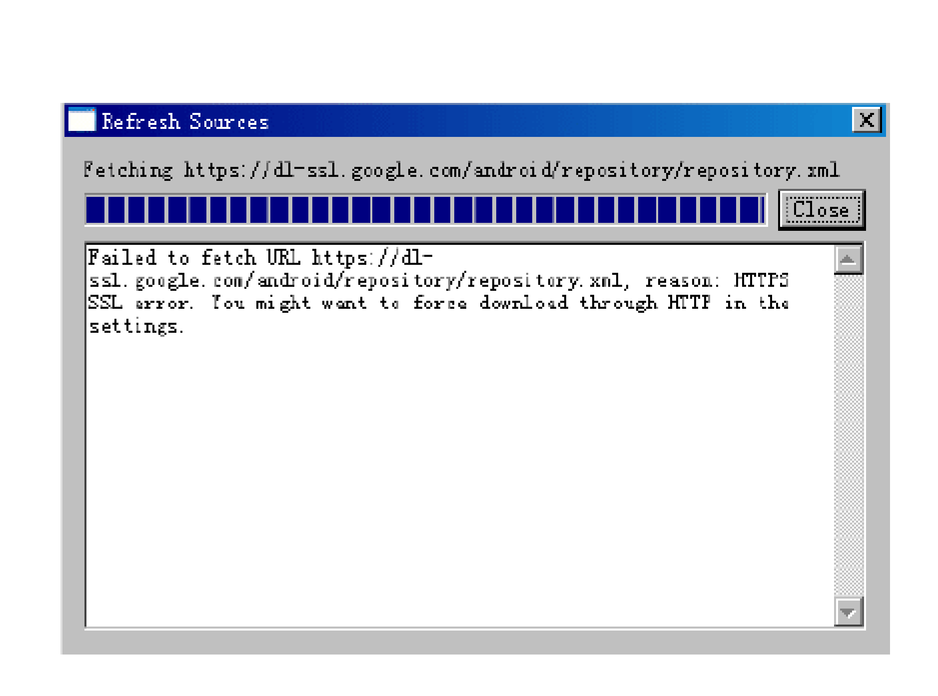


图 2-39　不能更新

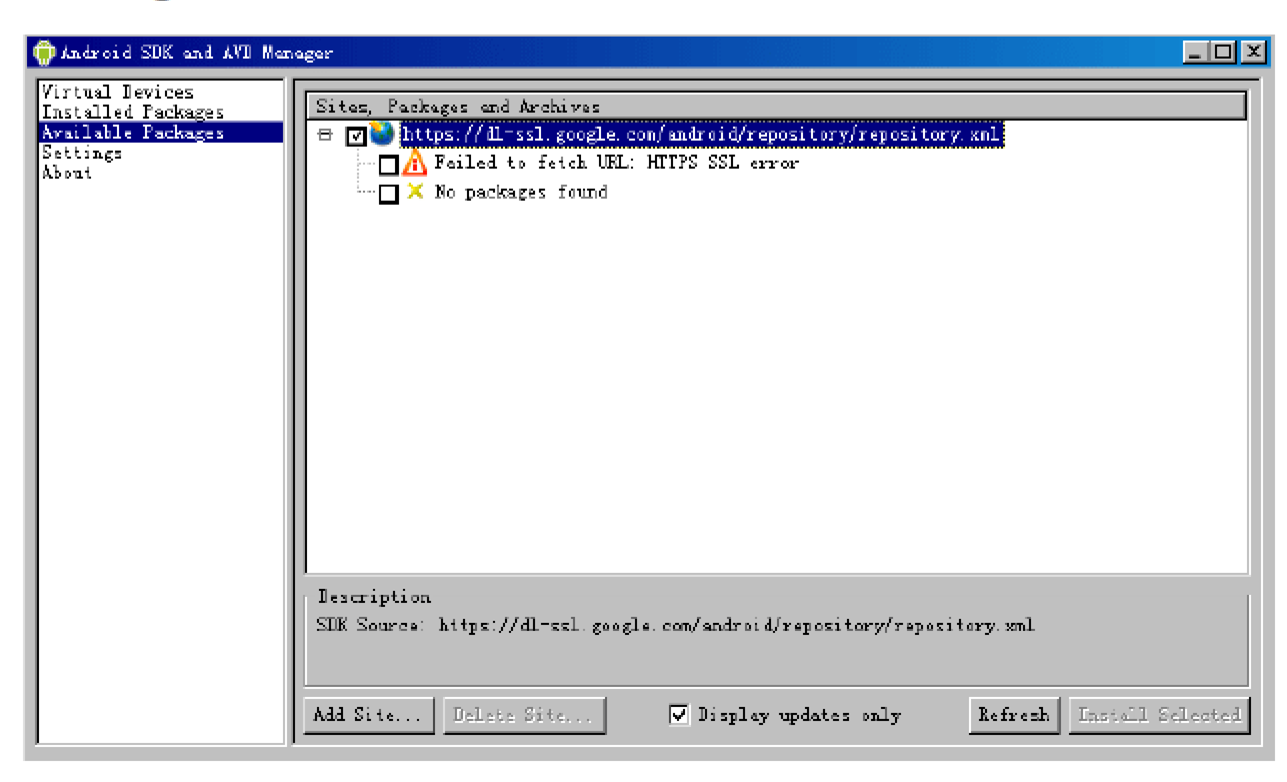


图 2-40　Available Packages 界面

(2)　在弹出的 Add Site URL 对话框中输入修改后的地址 http://dl-ssl.google.com/android/repository/repository.xml，如图 2-41 所示。

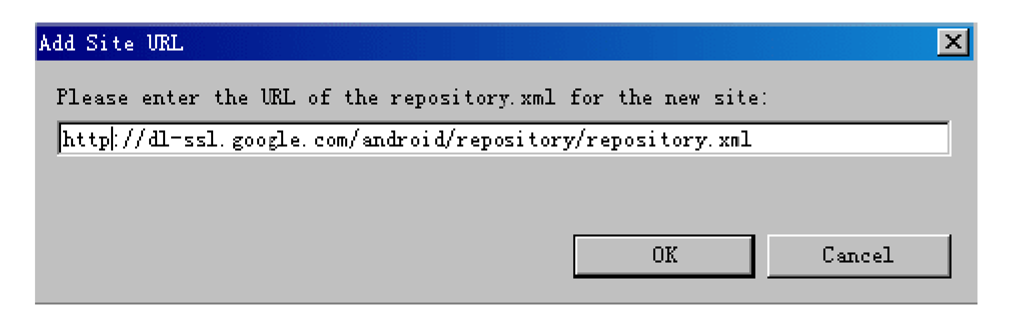


图 2-41　Add Site URL 对话框

(3)　单击 OK 按钮，完成设置。经过上述操作后，就能够使用更新功能了，如图 2-42 所示。

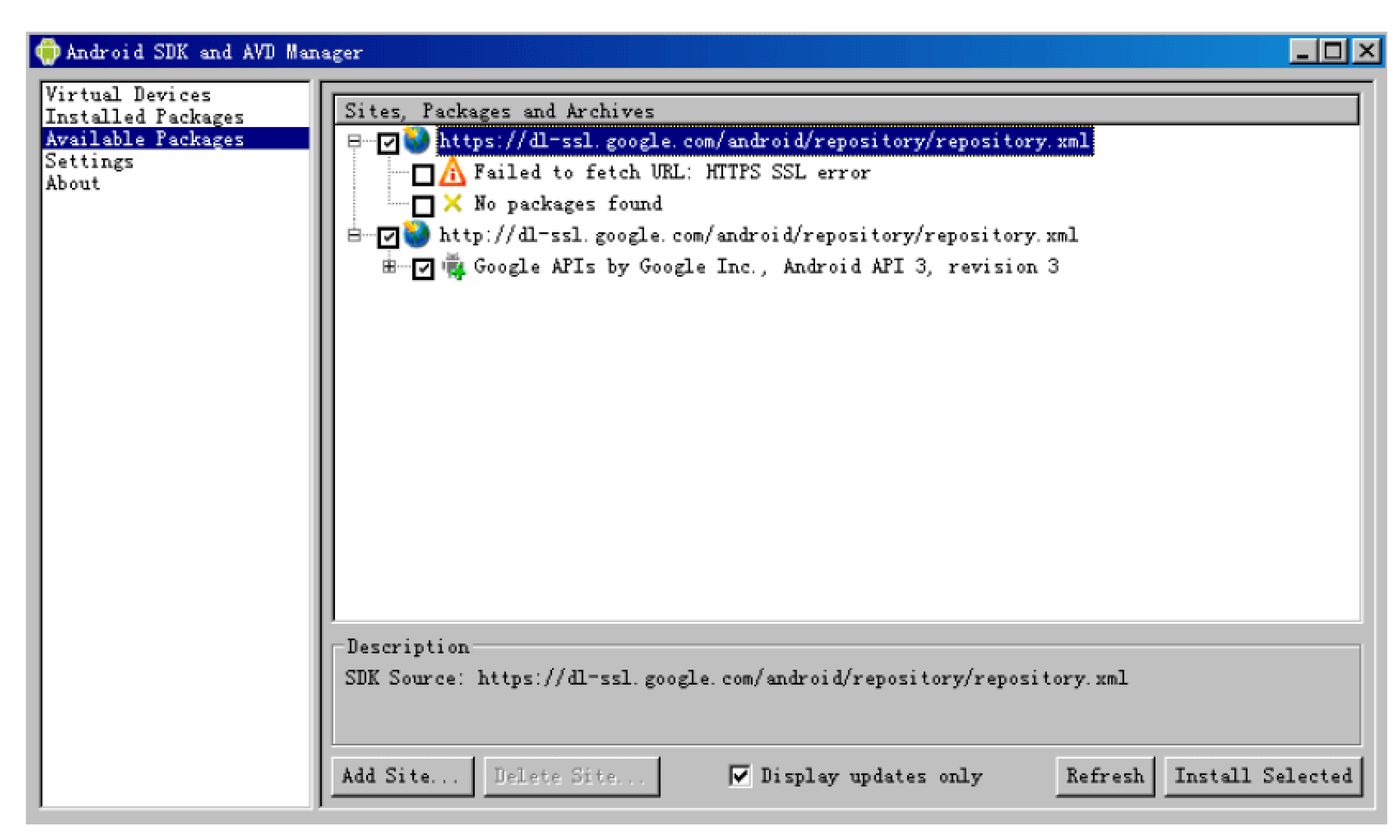


图 2-42　Available Packages 界面

2. Eclipse 中新建 Android 工程

Eclipse 中新建 Android 工程时，一直显示 Project name must be specified，如图 2-43 所示。造成上述问题的原因是 Android 没有更新完成，需要进行完全更新。具体操作方法如下。

(1) 打开 Android，选择左侧的 Installed Packages 选项，如图 2-44 所示。

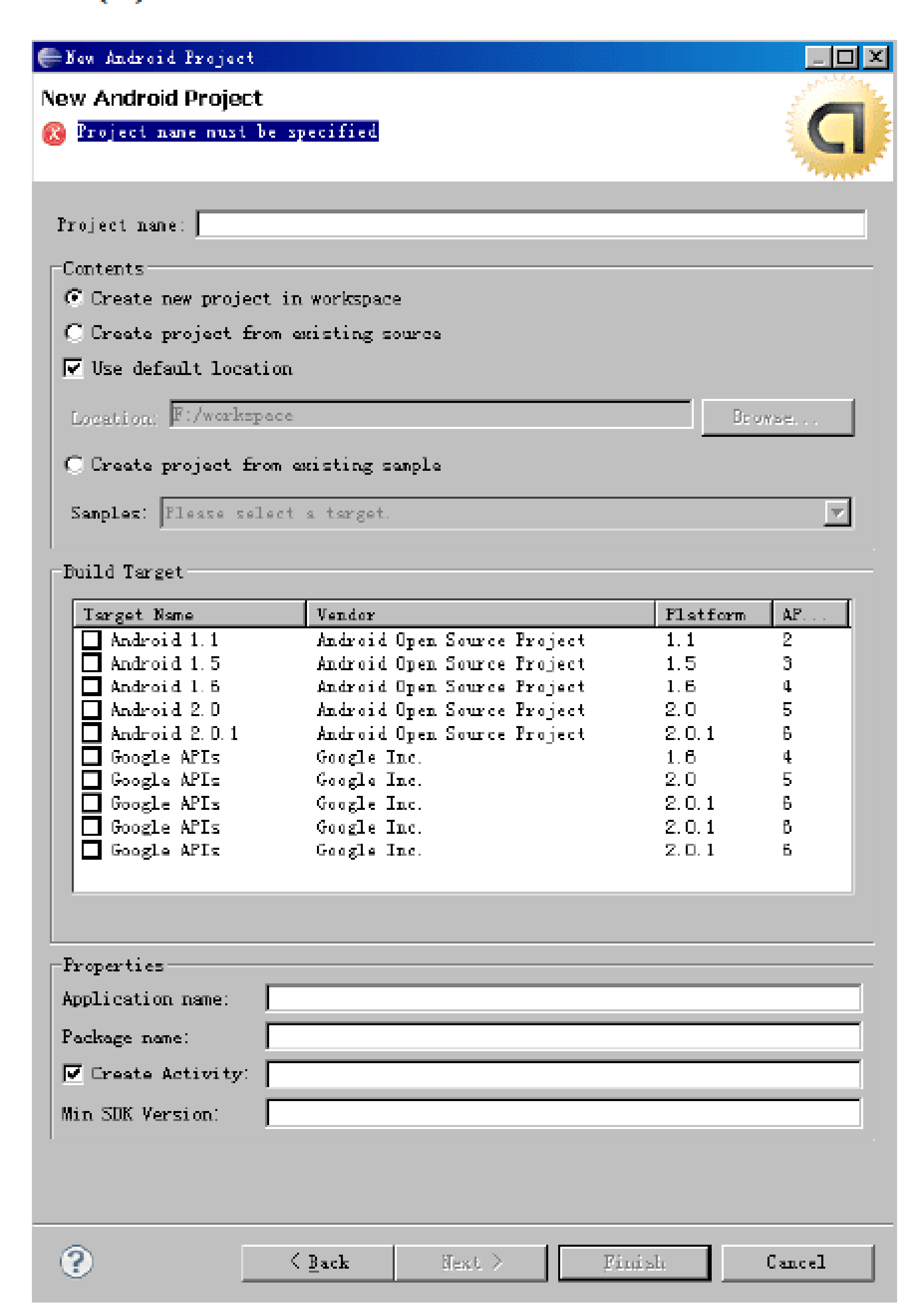


图 2-43 New Android Project 界面

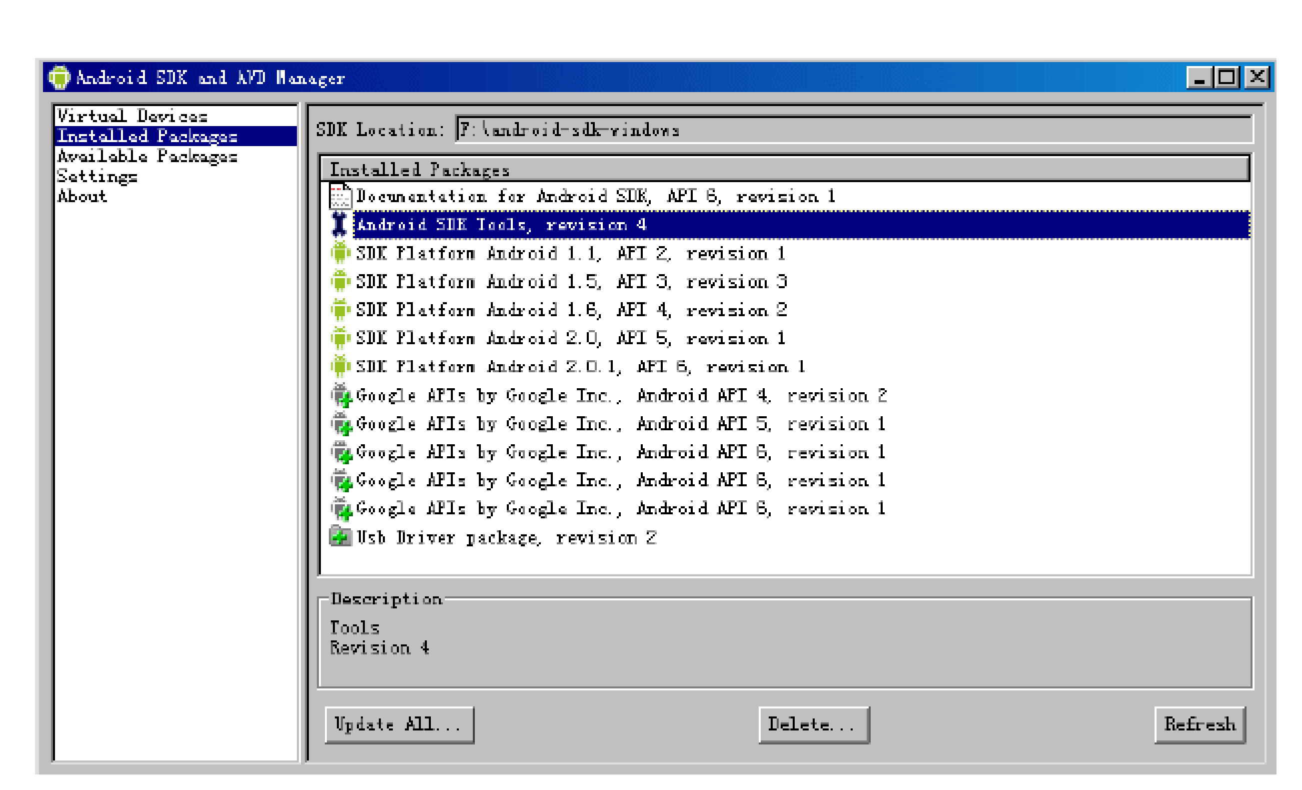


图 2-44 Installed Packages 界面

(2) 右侧列表中选择 Android SDK Tools, revision 4 选项，在弹出的窗口中选择 Accept 单选按钮，最后单击 Install Accepted 按钮后开始安装更新，如图 2-45 所示。

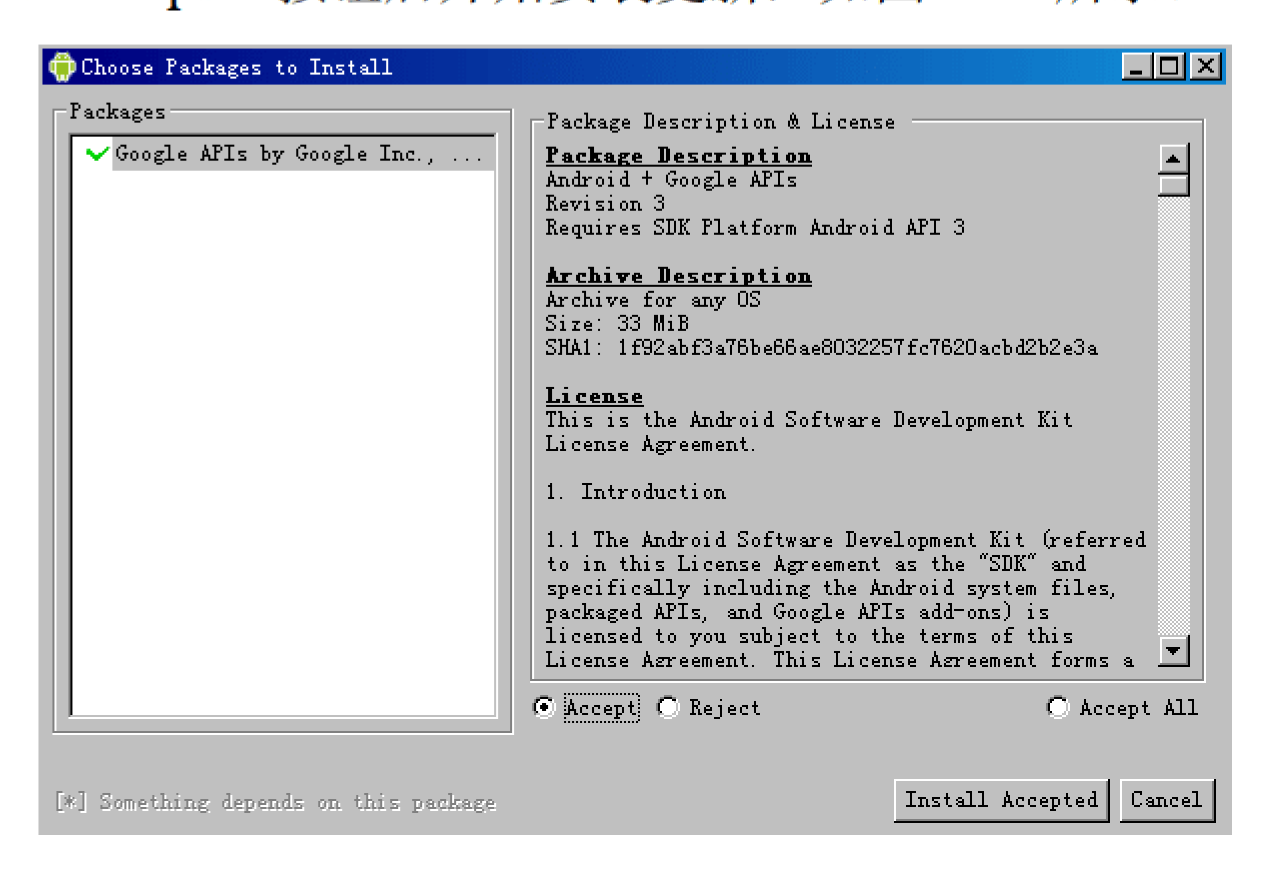


图 2-45 Choose Packages to Install

3. Target 列表中没有 Target 选项

通常来说，当 Android 开发环境搭建完毕后，在 Eclipse 中选择 Window | Preferences 命令，单击左侧的 Android 选项后，会在 Preferences 中显示存在的 SDK Targets，如图 2-46 所示。

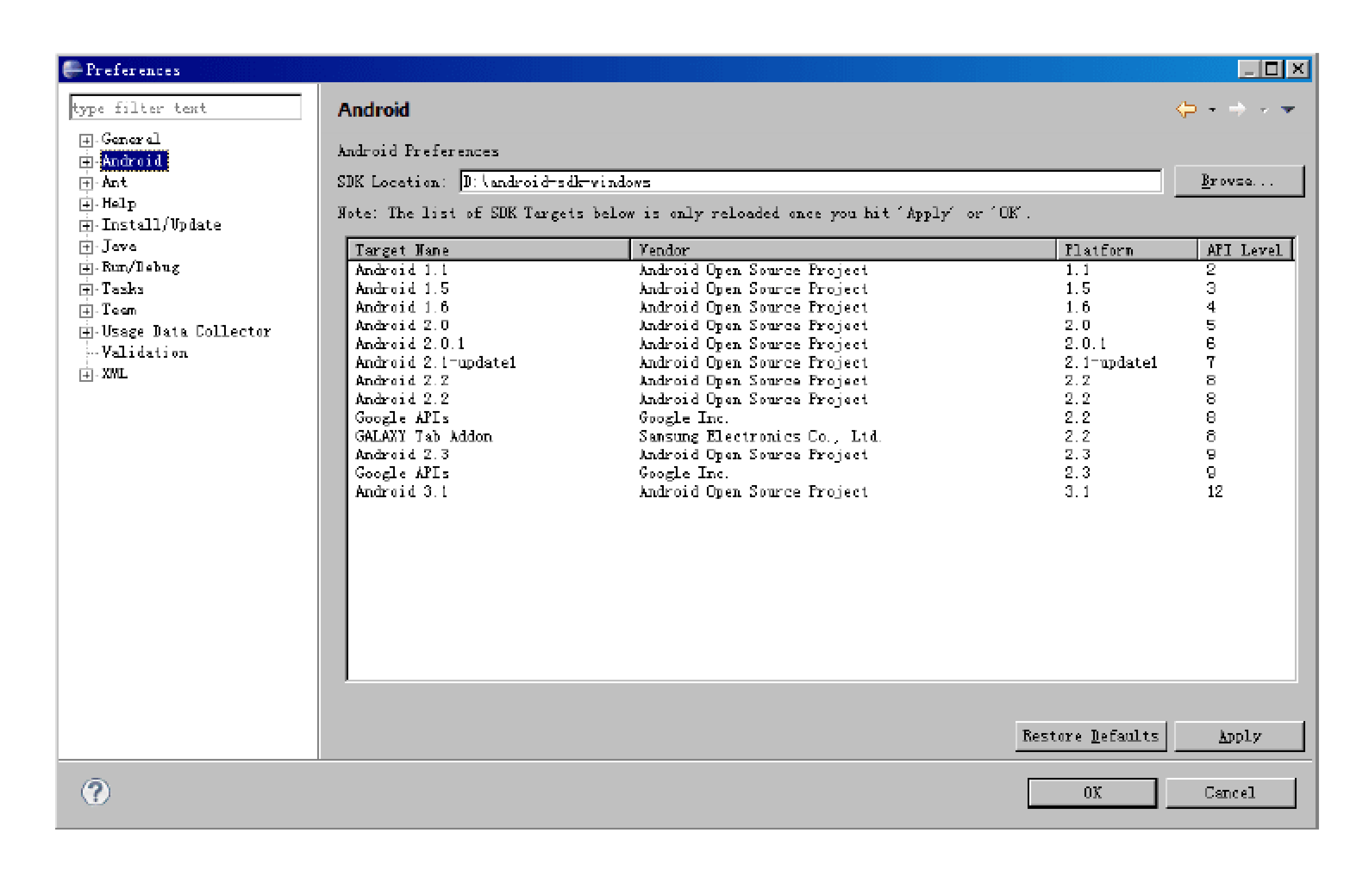


图 2-46　SDK Targets 列表

但是往往因为各种原因，会不显示 SDK Targets 列表，并且在图 2-32 所示的对话框中也不显示，而是输出“Failed to find an AVD compatible with target”提示的错误问题。上述问题是因为 AVD 没有创建成功，如果出现上述问题，就需要我们手工安装，当然前提是 Android 已经更新完毕。具体解决方法如下。

(1) 在“运行”对话框的“打开”文本框中输入“CMD”，打开 CMD 窗口，如图 2-47 所示。

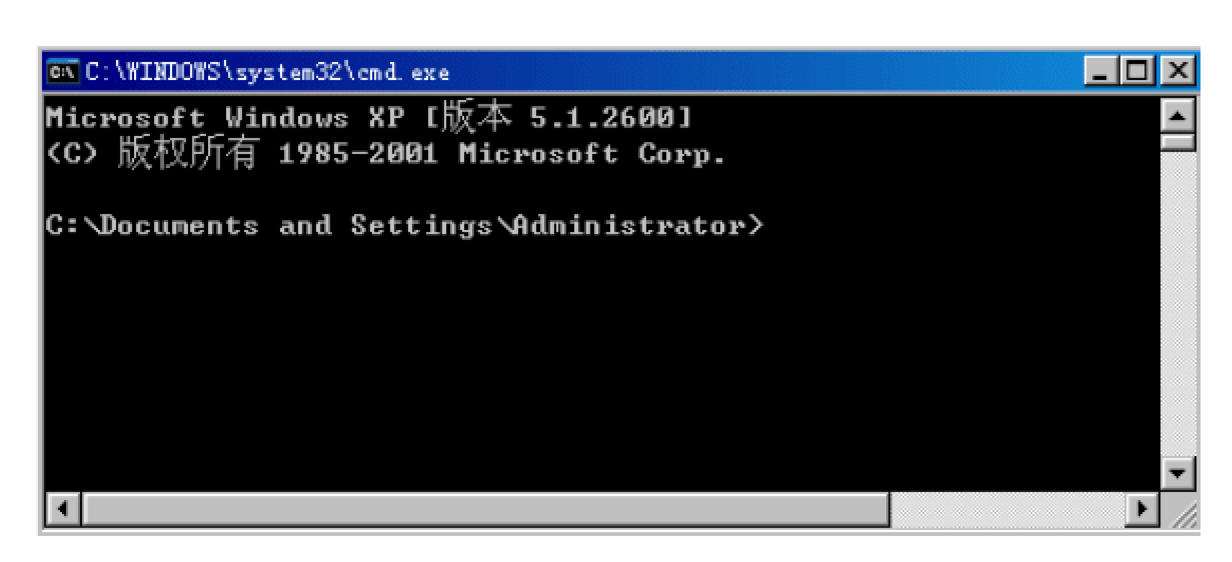


图 2-47　CMD 窗口

(2) 使用 android 命令创建一个 Targets，按照“android create avd --name <your_avd_name> --target <targetID>”格式创建 AVD，其中“your_avd_name”是需要创建的 AVD 的名字，在 CMD 窗口中，如图 2-48 所示。

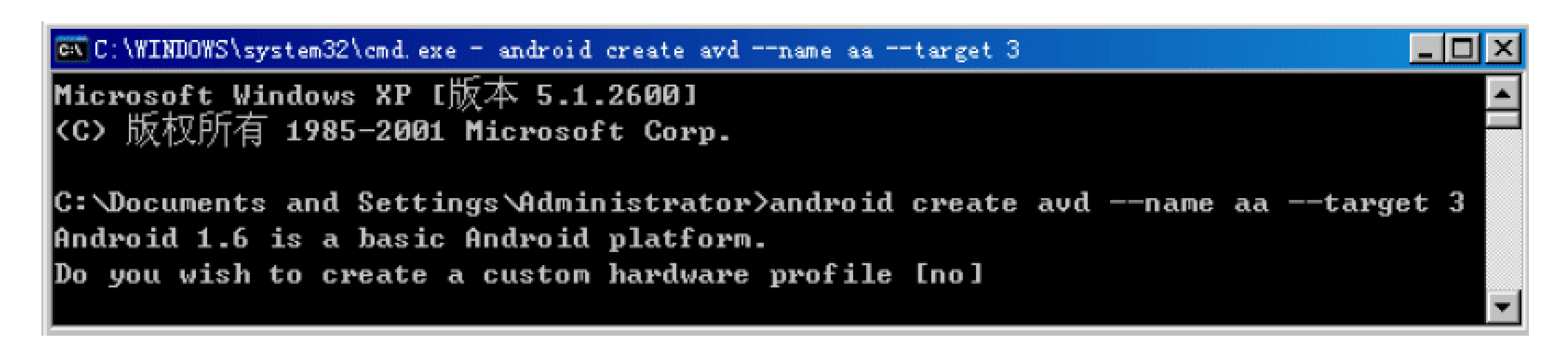


图 2-48　输入命令创建界面

(3) 图 2-48 所示的窗口中创建了一个名为“aa”，targetID 为“3”的 AVD，然后在 CMD 窗口中输入“n”，即可完成操作，如图 2-49 所示。

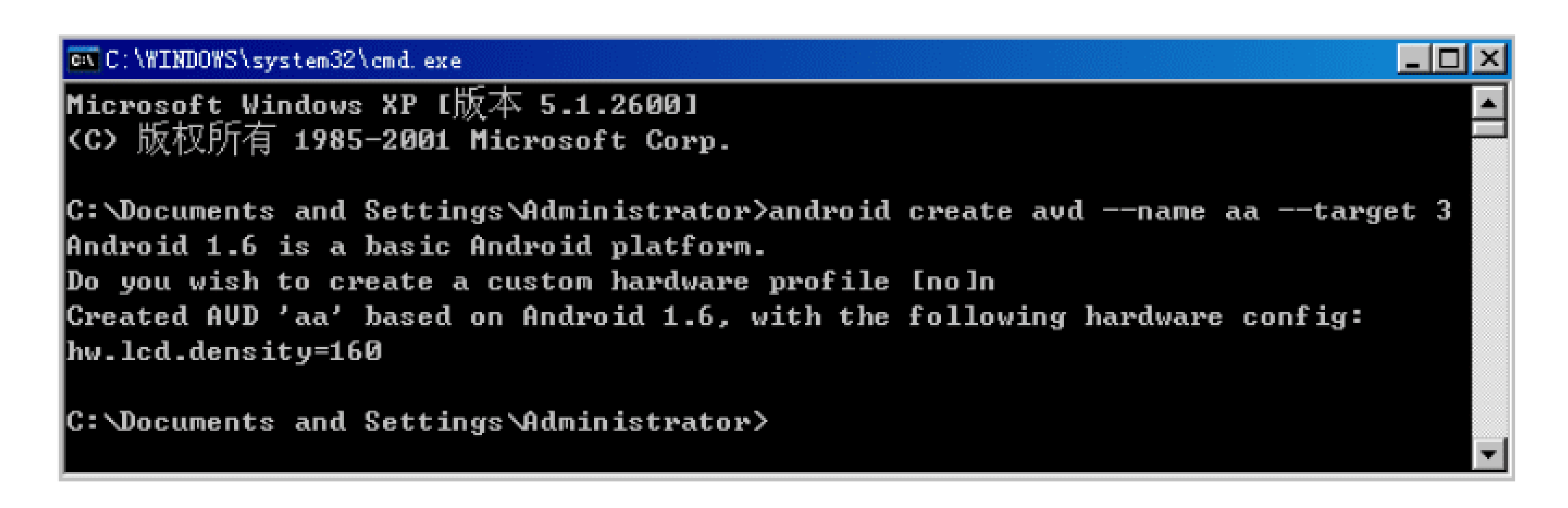


图 2-49　创建成功界面

2.5　Android 模拟器

Android 中提供了一个模拟器来模拟 ARM 核的移动设备。Android 的模拟器是基于 QEMU 开发的，QEMU 是一个有名的开源虚拟机项目(详见 http://bellard.org/qemu/)，它可以提供一个虚拟的 ARM 移动设备。开发人员不需要一个真实的手机，通过电脑模拟运行一部手机，即可开发出应用在手机上的程序。模拟器在电脑上模拟运行的效果如图 2-50 所示。



图 2-50　模拟器模拟手机

2.5.1　Android 模拟器基础

Android 模拟器被命名为 goldfish，用来模拟下面的功能：

- ARM926ej-S CPU；
- Thumb support；
- MMC；
- RTC；
- Keyboard；
- USB Gadget；
- framebuffer；
- TTY driver；
- NAND FLASH。

Android 模拟器所对应的源代码主要是在 external/qemu 目录下。如果你想将 Android 移植到其他设备上，熟悉它目前所针对的模拟器环境可以提供一些参考。

对于应用程序的开发者，模拟器提供了很多开发和测试时的便利。无论在 Windows 下还是 Linux 下，Android 模拟器都可以顺利运行，并且 Google 提供了 Eclipse 插件，可将模拟器集成到 Eclipse 的 IDE 环境。当然，你也可以从命令行启动 Android 模拟器。

这款模拟器功能非常齐全，电话本、通话等功能都可正常使用(当然你没办法真的从这里打电话)，甚至其内置的浏览器和 Google Maps 都可以联网。用户可以使用键盘输入，或者鼠标单击模拟器按键输入，甚至还可以使用鼠标单击、拖动屏幕进行操纵。

2.5.2　和实战的区别

Android 模拟器和真机的不同之处在于：

- 不支持呼叫和接听实际来电，但可以通过控制台模拟电话呼叫(呼入和呼出)；
- 不支持 USB 连接；
- 不支持相机/视频捕捉；
- 不支持音频输入(捕捉)，但支持输出(重放)；
- 不支持扩展耳机；
- 不能确定连接状态；
- 不能确定电池电量水平和交流充电状态；
- 不能确定 SD 卡的插入/弹出；
- 不支持蓝牙。

2.5.3　创建和启动模拟器

要使用 GPhone 的模拟器，需要先到 http://developer.Android.com/sdk(如果打不开就用 http://Androidappdocs.appspot.com/sdk/index.html)上下载 Android 的 SDK，解压出来后在 SDK 的根目录下有一个 tools 文件夹，里面就有模拟器和一些非常有用的工具。

要正确地启动模拟器，你必须先要创建一个 AVD(虚拟设备)，读者可以利用 AVD 创建基于不同版本的模拟器，下面就介绍如何创建 AVD。

(1)　查看当前支持版本(在列出的版本中我们需要记住 id 值，这个值在第 2 步中使用)：

```
magicyu@magicyu-desktop:~$ Android list target
```

你可以看到几个 Available Android targets，比如 Name: Android 1.6，它们有各自的 id 号。

(2)　创建 AVD，代码如下：

```
magicyu@magicyu-desktop:~$ Android create avd -n magicyu -t 2
```

-n 后面接需要创建 AVD 的名字，-t 后面接需要创建虚拟器的类型，2 即为步骤(1)中得到的类型 id 号。

(3)　查看是否创建成功(如果成功会显示刚才我们创建的 AVD 信息)：

```
magicyu@magicyu-desktop:~$ Android list avd
```

(4)　启动模拟器，代码如下：

```
magicyu@magicyu-desktop:~$ emulator @magicyu
```

或者

```
emulator -avd magicyu
```

其中@和-avd 后接的是你创建过的 AVD 名字。

(5)　选择启动的皮肤，代码如下：

```
magicyu@magicyu-desktop:~$ emulator -avd magicyu -skin QVGA
```

skin 后面接所要启动皮肤的类型，所有的类型可以在/platforms/Android-1.*/skins 目录下找

到，*为所指的版本。如在 1.6 版本的 SDK 下有 HVGA、QVGA、WVGA800、WVGA854 几种。那么按 Ctrl+F11 组合键可以直接改变模拟器的横竖摆放。

当然 AVD 也可以在 Eclipse 中创建和启动。

运行 Eclipse，依次选择 Window | Android SDK and AVD Manager 菜单命令，单击 Android SDK and AVD Manager 界面下方的 New 按钮可以新建一个 AVD，如图 2-51 所示。

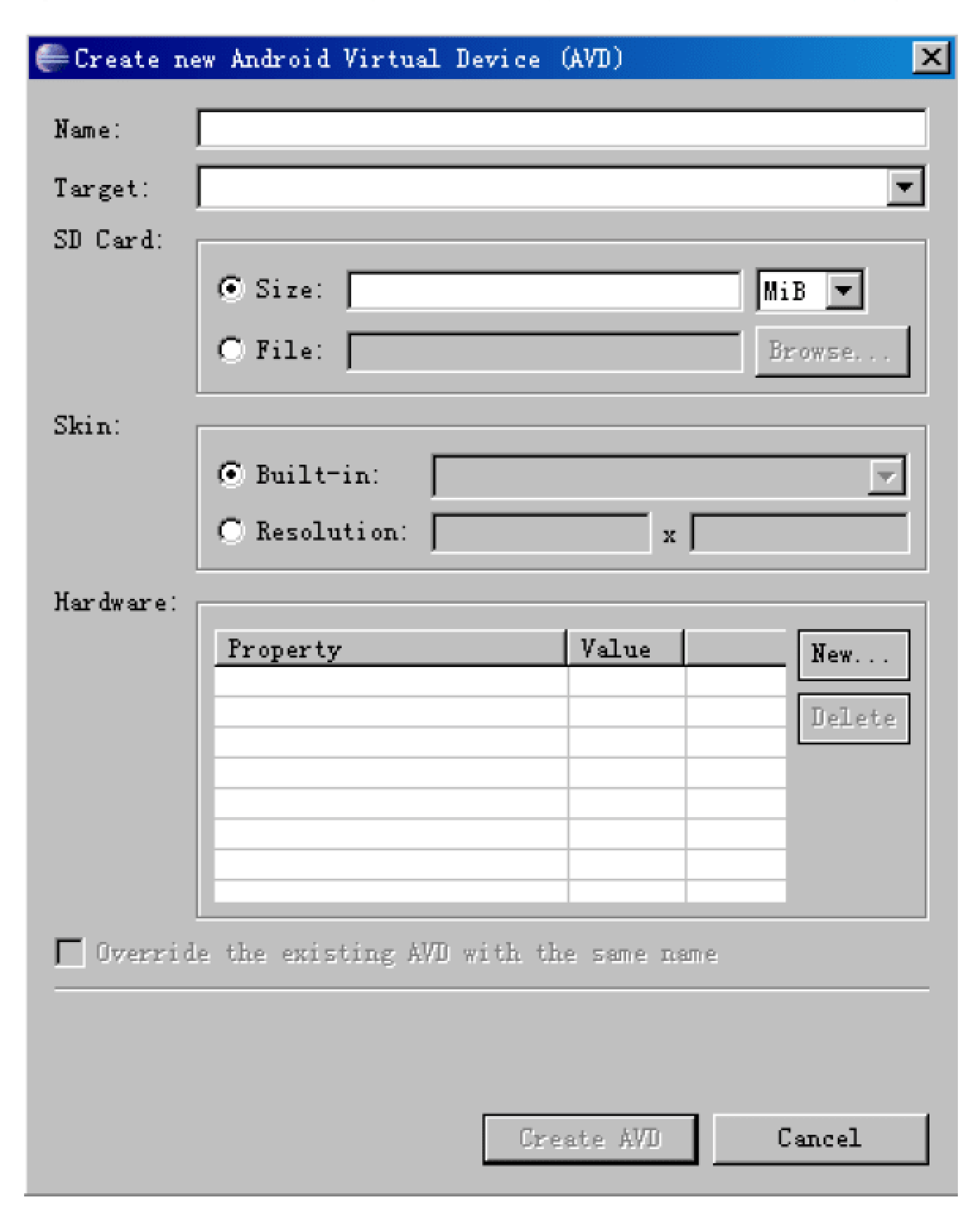


图 2-51　新建 AVD 界面

第 3 章　庖丁解牛 Android SDK

经过前面内容的学习，读者已经了解了 Android SDK 的重要作用和地位，并了解了其具体的安装流程。本章将进一步对 Android SDK 集成开发环境进行剖析，这样将便于读者理解本书后面实例的开发过程。

3.1　得心应手是第一要务

自从上山学艺之后，我每天都是闻鸡起舞，立志早日成为一代侠客。每天早晨、上午习武各两个时辰，中午用一个时辰的时间听师傅传授武功心法。

你们手上的 Android SDK 是一个好东西啊，它以 Java 语言为基础，你们可以使用 Java 来开发 Android 平台上的应用软件。通过 SDK 提供的一些工具将其打包成 Android 平台使用的 apk 文件，然后再使用 SDK 中的模拟器来模拟和测试该软件在 Android 上的运行效果。

想必大家都听说过“庖丁解牛”这个故事，它出自《庄子》，比喻经过反复实践，掌握了事物的客观规律，做事得心应手、运用自如。今天我所要传授的武功心法是 Android SDK。没错，就是你们已经装备了的 Android SDK。我们现在已经有了绝世好剑，但是你们了解它吗？你们必须详细剖析它的方方面面，做到得心应手。你们要具备庖丁解牛的精神，将手中的 Android SDK 全面剖析，做到能够随心所欲地使用！

3.2　初步探寻 Android SDK 体系

当我打开安装后的 Android，发现在其安装目录中有很多安装文件。这些文件究竟是干什么用的呢？我决定好好分析一番。仔细看着我手中的绝世好剑，我觉得应该从图 3-1 所示的领域下手分解。

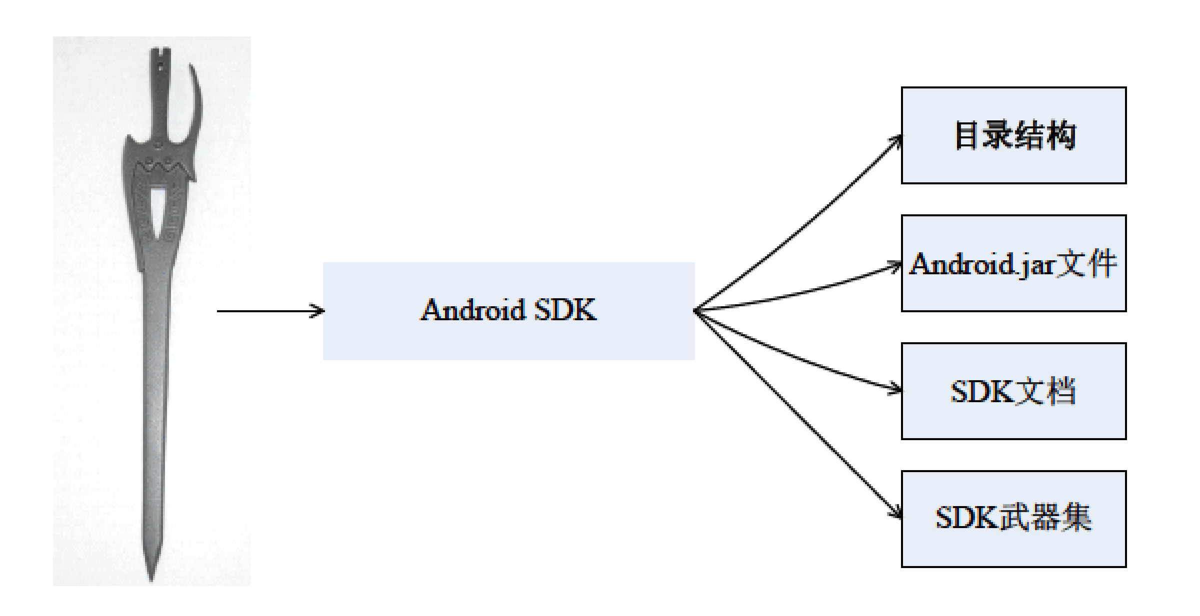


图 3-1　剖析 Android SDK 3.1 的领域

3.2.1　目录结构

分析目录结构是最直观的，我打开 Android SDK 的安装目录，发现其目录结构如图 3-2 所示。

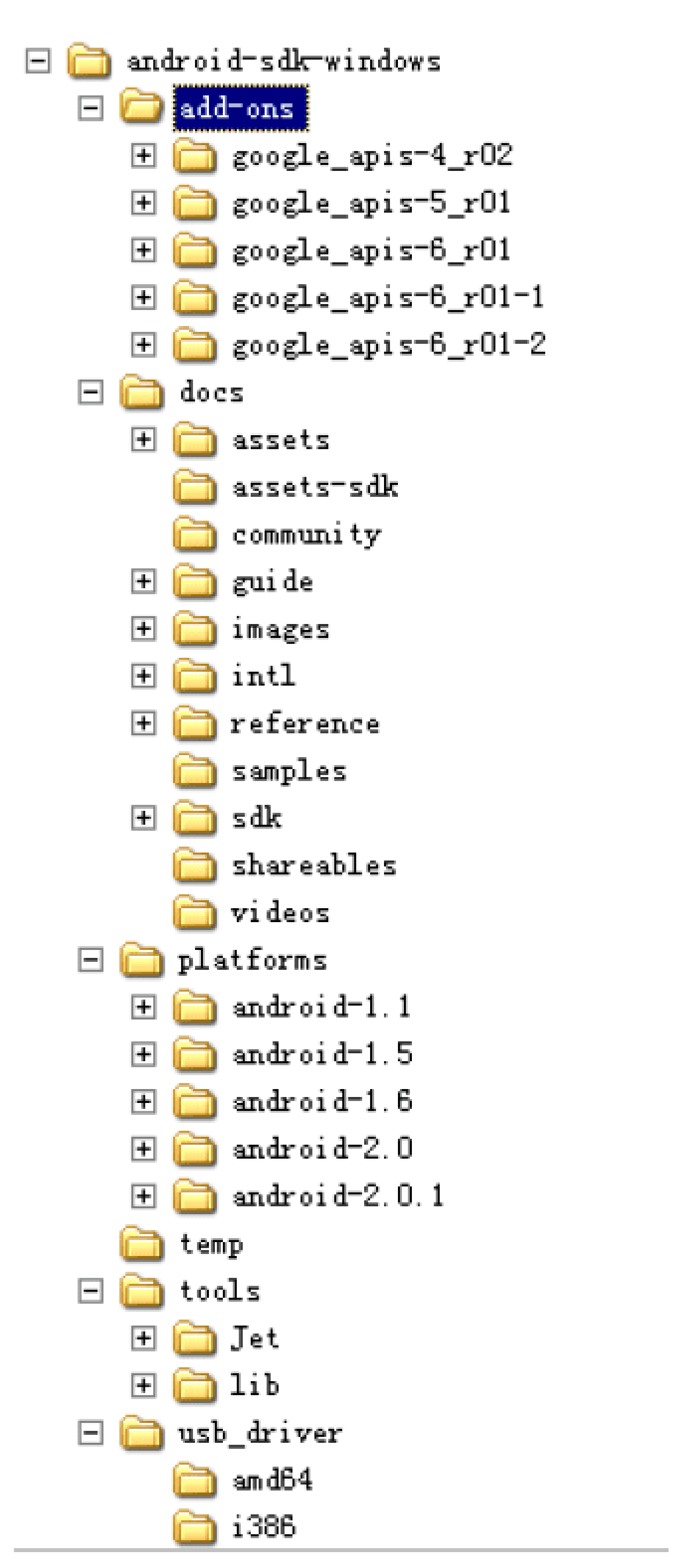


图 3-2　Android SDK 安装后的目录结构

- add-ons：包含了 google 提供的 API 包，例如最常见的 google Map 就属于众多 API 中的一种。
- docs：包含了帮助文档和说明文档。
- platforms：针对每个版本的 SDK 提供了和其对应的 API 包以及一些示例文件，其中包含了各个版本的 Android，如图 3-3 所示。
- temp：包含了一些常用的文件模板。
- tools：包含了一些通用的工具文件。

- usb_driver：包含了 AMD64 和 x86 下的驱动文件。

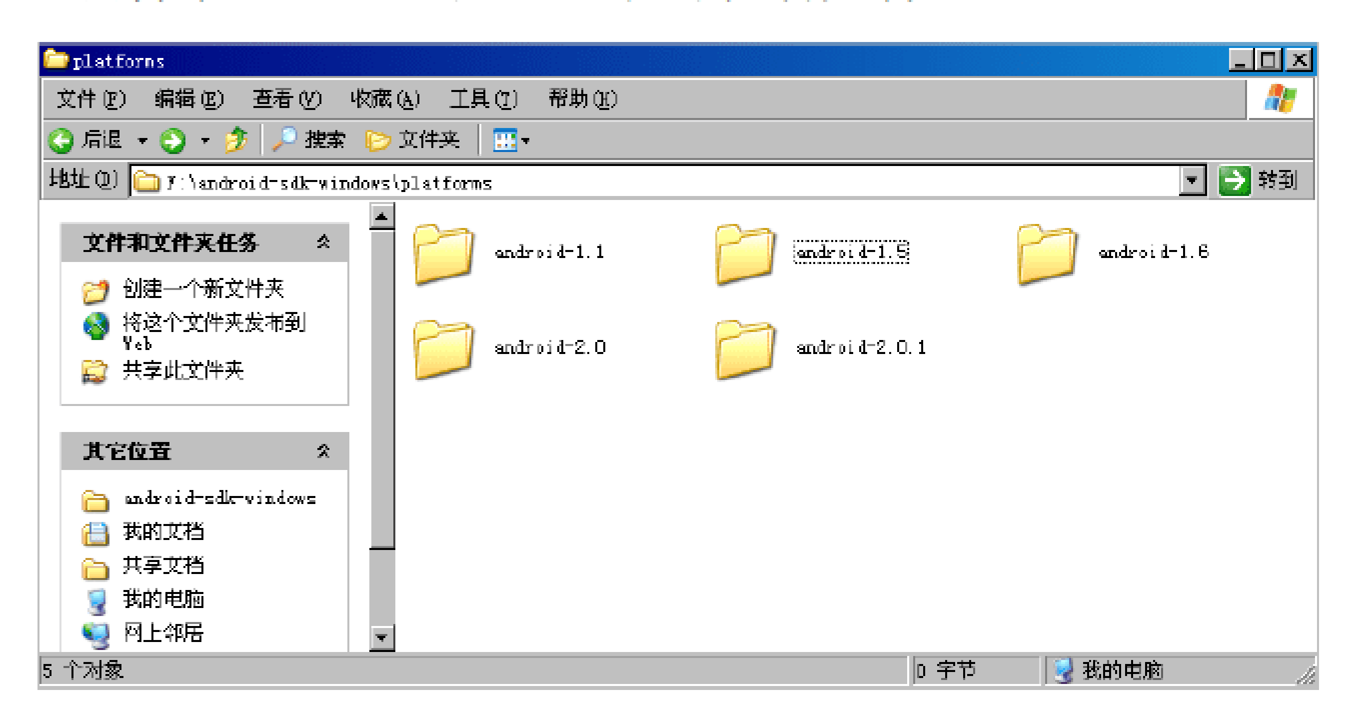


图 3-3　platforms 目录项

3.2.2　解剖 Android.jar

目录中最核心的是 platforms 文件夹，在 platforms 目录下的每个 Android 版本的文件夹中，都有一个 Android.jar 文件。例如图 3-4 所示的 Android-2.0 文件夹。

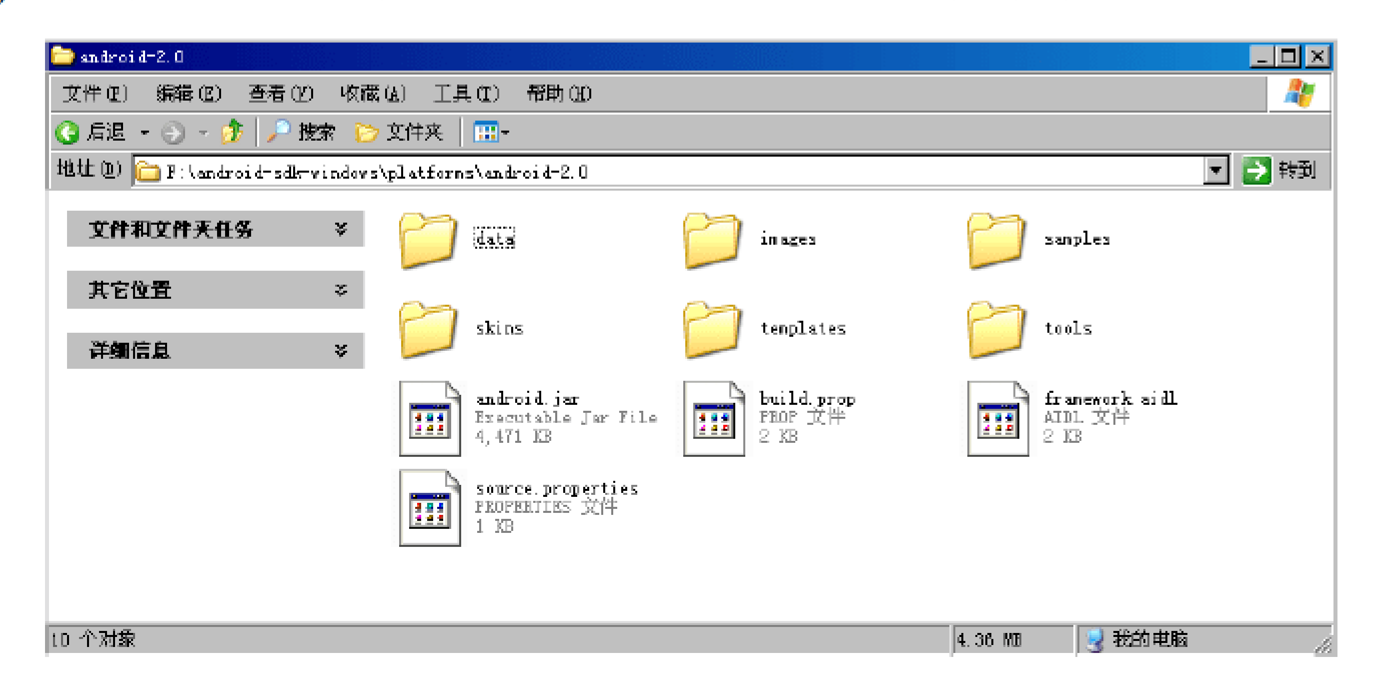


图 3-4　Android.jar 文件所在目录

Android.jar 是一个标准的压缩包，里面包含了编译后的压缩文件，包含了全部的 API，使用 Windows 系统上的解压缩工具 WinRAR 可以打开此压缩文件，此时可以看到其内部结构，如图 3-5 和图 3-6 所示。

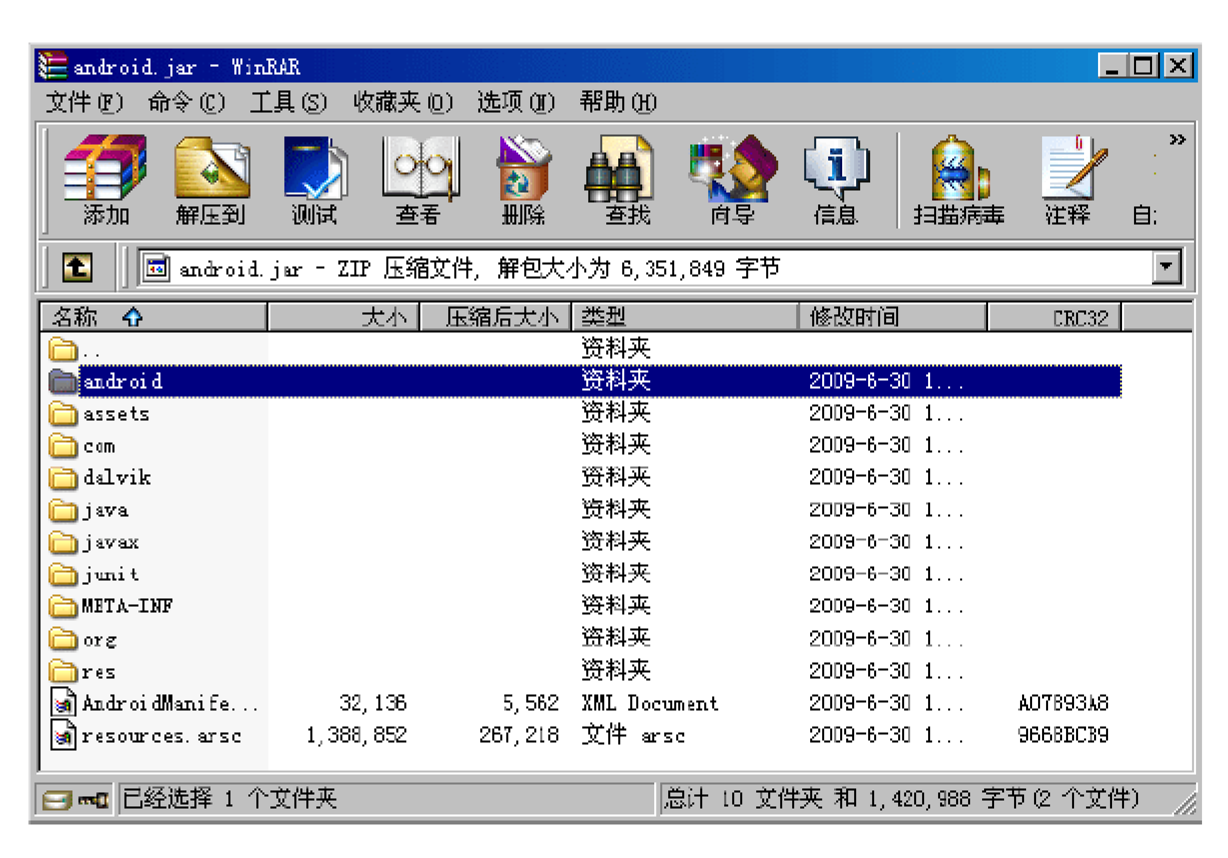


图 3-5　Android.jar 文件结构

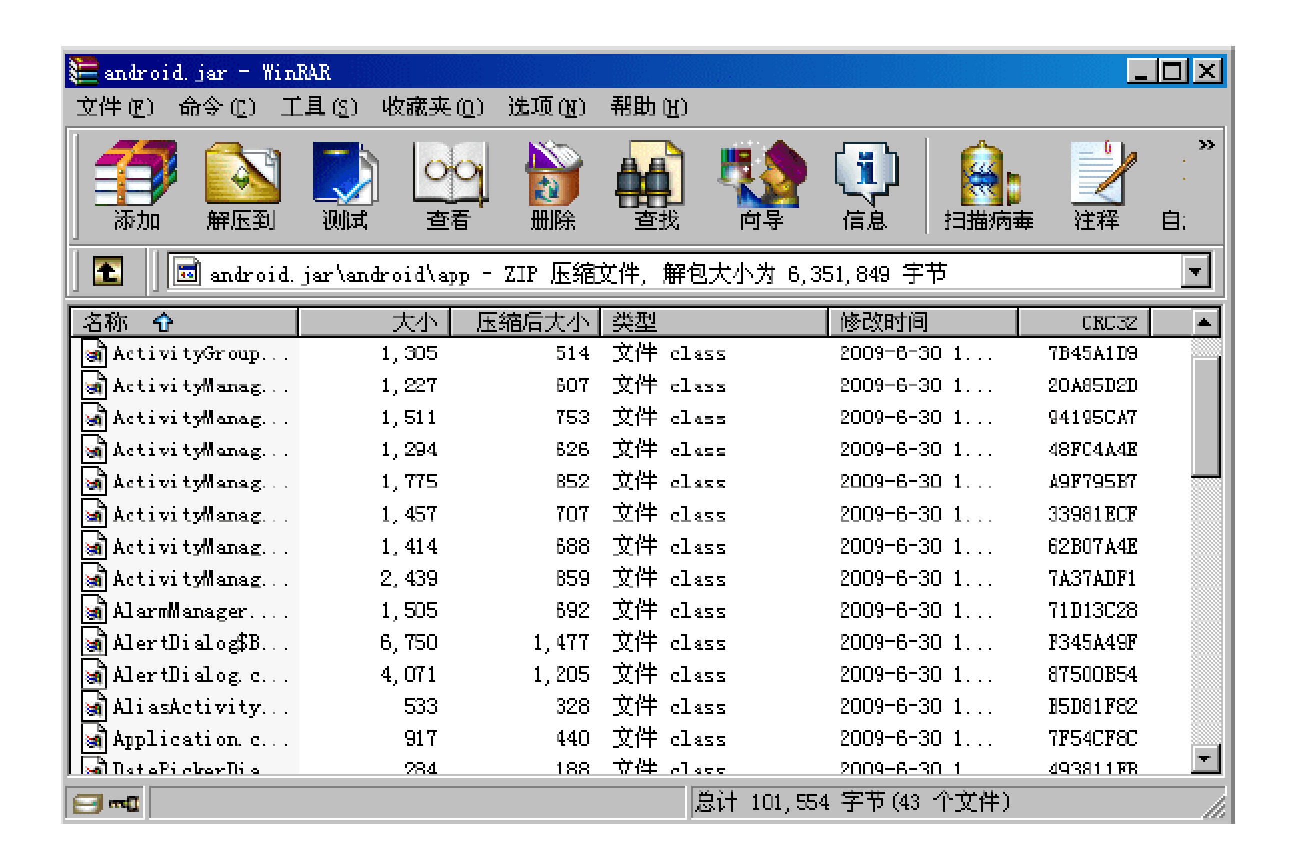


图 3-6 Android.jar 文件结构

从图 3-5 和图 3-6 所示的结构可以看出，它对 API 包进行了详细的划分，分为了 app、content、database 等。只要我们理解了其模块划分结构，就可以通过 SDK 文档了解其具体信息。

3.2.3 SDK 文档是你的良师

通过解压缩 Android.jar 后，我了解了其内部 API 的包结构和组织方式。我决定继续发扬庖丁解牛的精神，进一步深入理解各个文件包内包含的 API 和 API 的具体用法。

我用浏览器打开“docs”目录下的 index.html，如图 3-7 所示。

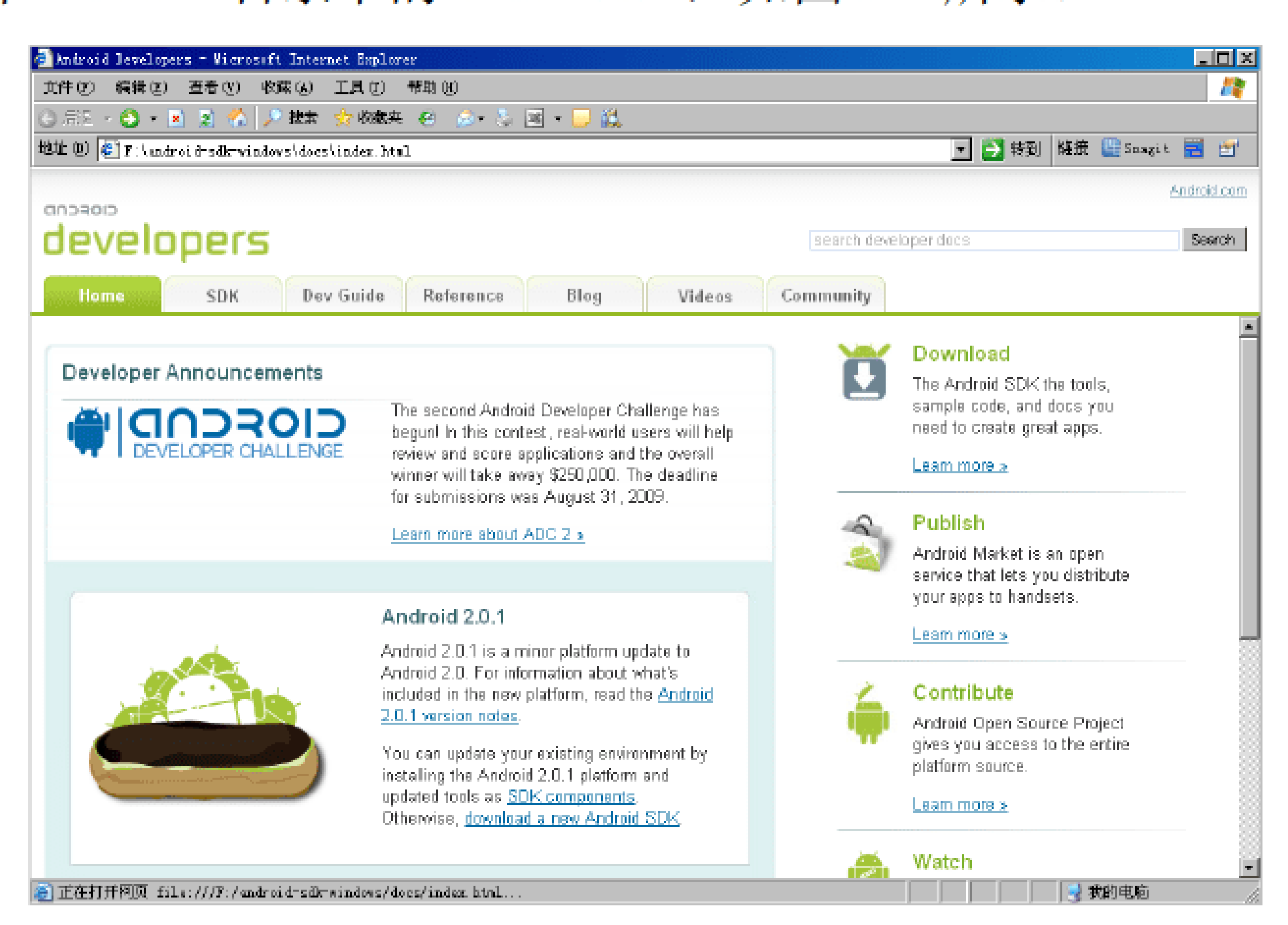


图 3-7 SDK 文档主页

在图 3-7 所示的主页中，显示了 Android 的基本概念、当前版本，在右侧和顶端导航中列出了一些常用的链接。这个 SDK 文件真是太重要了，它介绍了 Android 的基本知识，是一个很好的学习文档、帮助文档，能帮助我解决很多常见的问题。

单击导航栏中的 Dev Guide 按钮，打开 SDK 文档索引页面如图 3-8 所示。在图 3-8 所示的页面中，左侧是目录索引链接，当单击某个链接后，可以在右侧页面中显示对应的说明信息。如果想迅速掌握一个问题或知识点，可以在搜索对话框中对 SDK 进行检索，搜索到自己需要的内容。

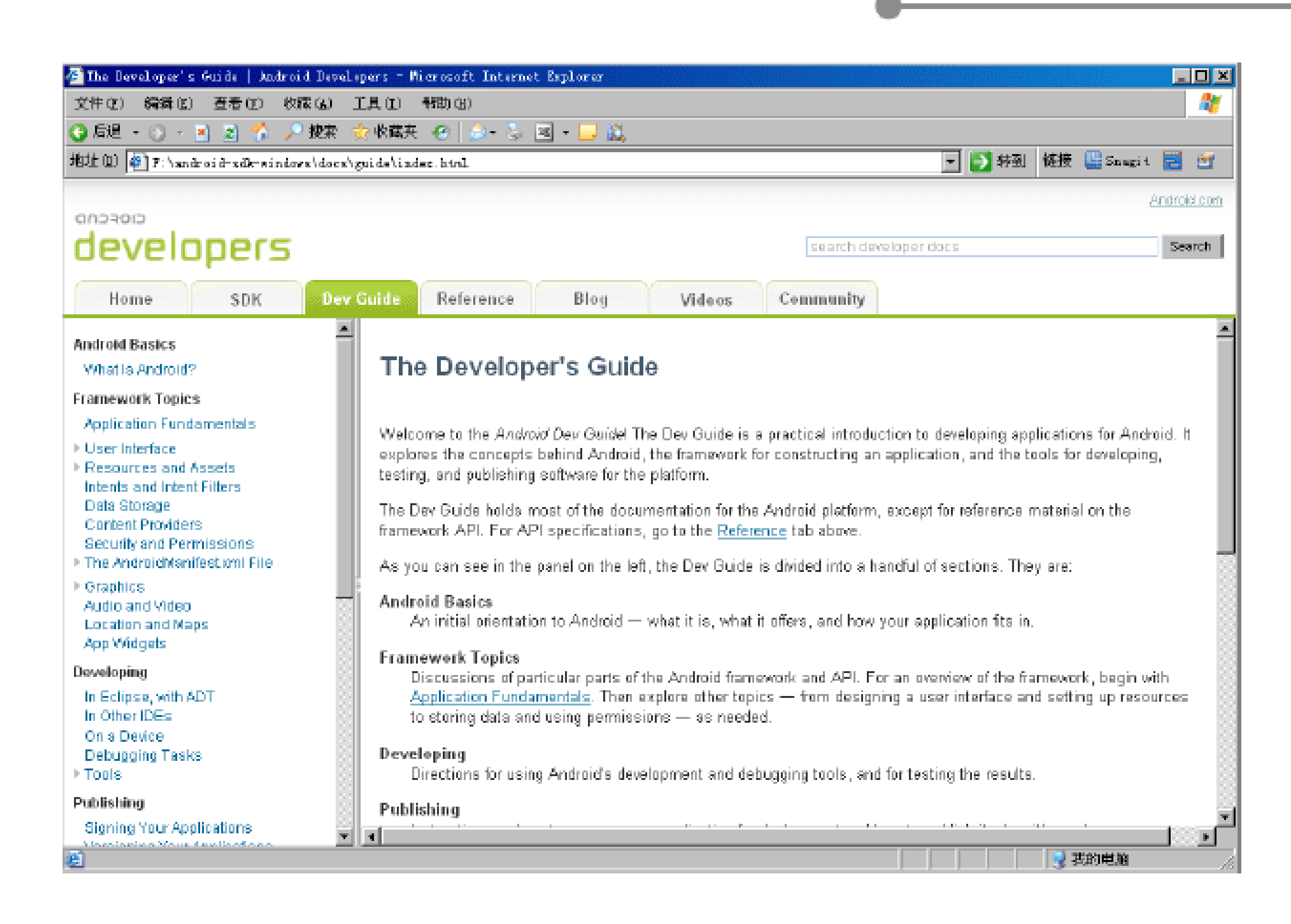


图 3-8　SDK 文档索引

3.2.4　SDK 武器集

前面的内容只能算是武功秘籍，其实 SDK 的功能何止这些，它还为我们提供了各种武器，这些武器能够帮助用户在 Android 平台上开发出有用的应用程序。在 Android SDK 中，最为常用的武器是 Android 模拟器和 Eclipse 的 Android 开发插件。其实，在 SDK 中也包含了各种在模拟器上用于调试、打包和安装的武器，具体如下。

1)　Android 模拟器

Android 模拟器即 AVD，是运行在计算机上的虚拟移动设备。

2)　集成开发插件(ADT)

Android 为 Eclipse 定制了集成开发插件(Android Development Tools，ADT)插件，这个插件为用户提供了一个强大的综合环境，用于开发 Android 应用程序。ADT 扩展了 Eclipse 的功能，可以让用户快速地建立 Android 项目，创建应用程序界面，在基于 Android 框架 API 的基础上添加组件，以及用 SDK 工具集调试应用程序，甚至导出签名(或未签名)的 APKs，以便发行应用程序。

3)　调试监视服务(ddms.bat)

调试监视服务 ddms.bat 集成在 Dalvik(Android 平台的虚拟机)中，用于管理运行在模拟器或设备上的进程，并协助调试工作。它可以去除一些进程，选择一个特定的程序来调试，生成跟踪数据、查看堆和线程数据，对模拟器或设备进行屏幕快照等操作。

4)　Android 调试桥(adb.exe)

Android 调试桥(adb)是多种用途的工具，该工具可以帮助用户管理设备或模拟器的状态。

可以通过下列几种方法加入 adb。

- ❑　在设备上运行 shell 命令。
- ❑　通过端口转发来管理模拟器或设备。
- ❑　从模拟器或设备上拷贝文件。

5)　Android 资源打包工具(aapt.exe)

可以通过 Android 资源打包工具(aapt.exe)来创建 apk 文件，在 apk 文件中包含了 Android 应用程序的二进制文件和资源文件。

6)　Android 接口描述语言 aidl.exe

Android 接口描述语言 aidl.exe 用于生成进程间的接口代码。

7)　SQLite3 数据库 sqlite3.exe

Android 可以创建和使用 SQLite 数据文件，开发人员和用户喜欢方便地访问这些 SQLite 数据文件。

8)　跟踪显示工具

跟踪显示工具可以生成跟踪日志数据的图形分析视图，这些跟踪日志数据由 Android 应用程序产生。

9)　创建 SD 卡工具

创建 SD 卡工具用于创建磁盘镜像，此镜像可以在模拟器上模拟外部存储卡，例如常见的 SD 卡。

10)　DX 工具(dx.bat)

DX 工具用于将 class 字节码重写为 Android 字节码(被存储在 dex 文件中)。

11)　生成 Ant 构建文件(activitycreator.bat)

activitycreator.bat 是一个脚本，用于生成 Ant 构建文件。Ant 构建文件用于编译 Android 应用程序，如果在安装 ADT 插件的 Eclipse 环境下进行开发，则不需要这个脚本。

12)　Android 虚拟设备

在 Android SDK1.5 版以后的 Android 开发中，必须创建至少一个 AVD，AVD 全称为 Android Virtual Device(Android 虚拟设备)，每个 AVD 模拟了一套虚拟设备来运行 Android 平台，这个平台至少要有自己的内核、系统图像和数据分区，还可以有自己的 SD 卡和用户数据以及外观显示等。

经过几个小时的“解牛”训练，我体会到了 Android SDK 的强大功能，它的文档如同武功秘籍，告诉我怎样使用 SDK。为了帮助用户轻松实现开发，领略智能手机的强大功能，它还提供了很多个辅助武器，帮助用户快速提高生产力。

3.3　师兄们的杰作

安卓派真是人才辈出，短短 3 年间诞生了 50 多名剑客。在安装目录中的“samples”文件夹内，记录了他们的英雄事迹。看着他们的作品我热血澎湃，恨不能马上下山去闯一番……

在“samples”的文件夹中存放了 SDK 中的几个使用实例，这些实例从不同的方面展示了 SDK 的特性。例如，图 3-9 所示的 Android-1.5 中的实例文件夹。

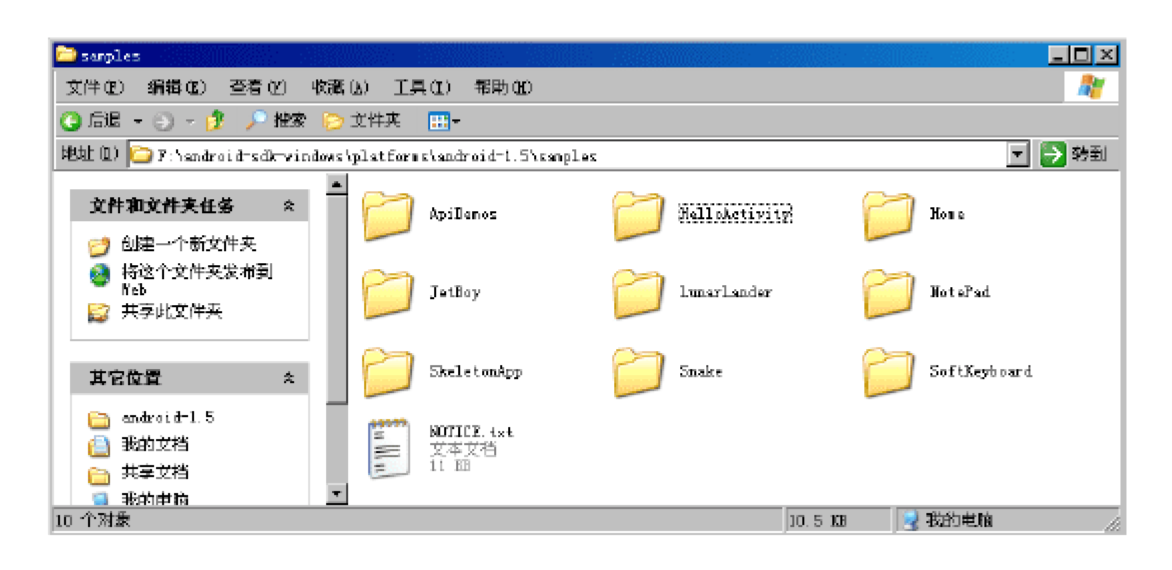


图 3-9　实例文件夹

1. HelloActivity

HelloActivity 或编程语言中的 Hello Word 程序类似，是 Android 平台上的一个最简单程序，运行后将在手机上显示出“Hello，World！”提示。打开 Eclipse，将“HelloActivity”导入，然后查看执行后的效果，具体如图 3-10 所示。

在查看安装目录中的“samples”实例时，不能使用“Import”将实例导入到 Eclipse 中。要查看实例的运行效果，需要按照下面的步骤操作。

(1)　在 Eclipse 中依次选择 file | new | Android project 菜单命令，在弹出的 New Android Project 对话框。选择 Create project from existing source 单选按钮，然后单击 Browse 按钮，选择对应的实例文件夹即可，如图 3-11 所示。

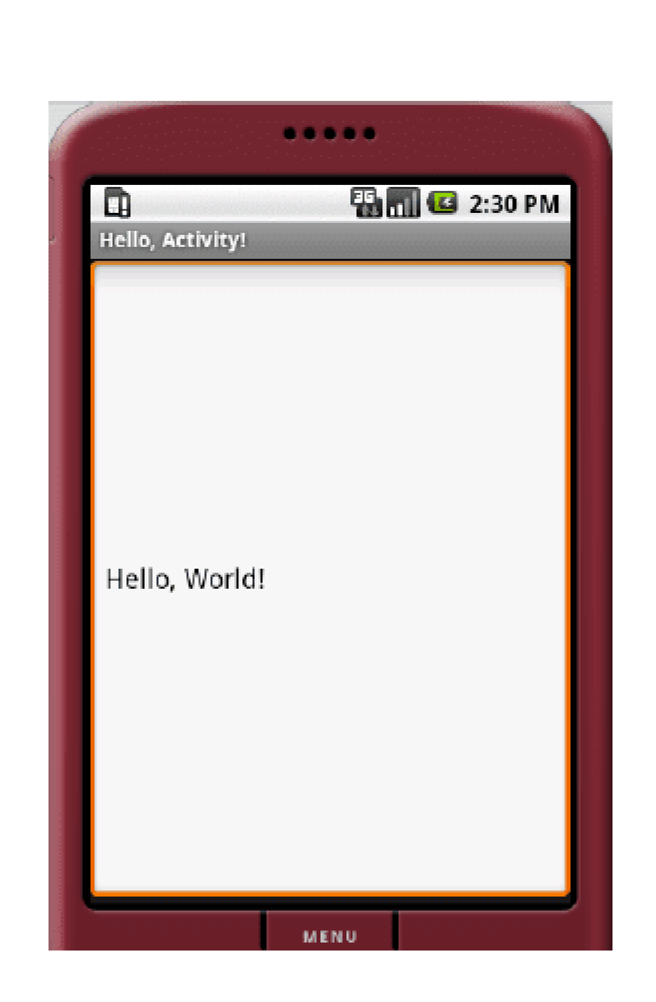


图 3-10　执行效果

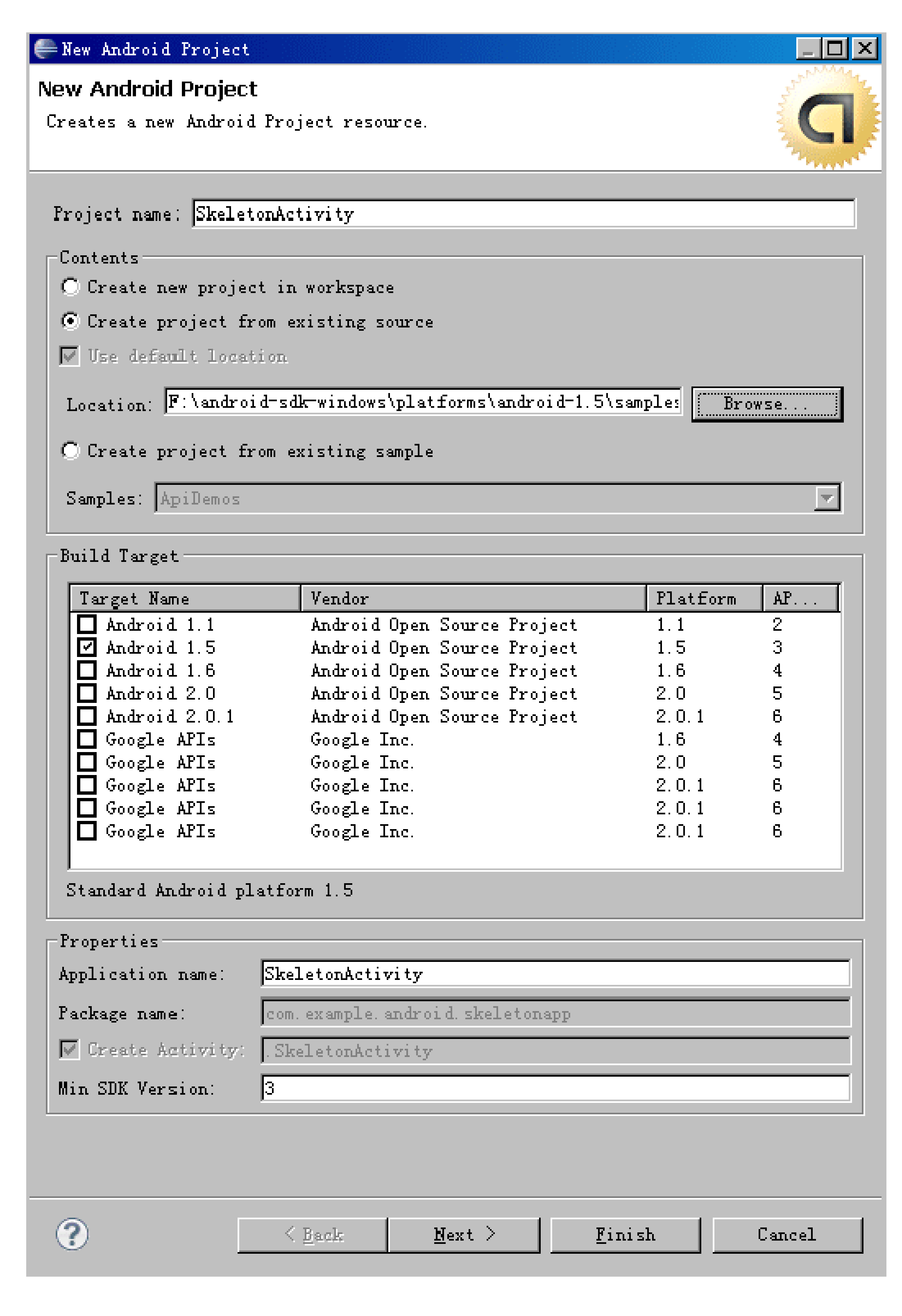


图 3-11　New Android Project 对话框

(2)　单击 Finish 按钮完成操作，这样即可将实例程序成功地导入到 Eclipse 中。

2. 视图组件 SkeletonApp

本实例展示了如何在 Android 中应用提供的视图组件，例如常见的 Edit Rext、Button、ImageView 和菜单等，并且还演示了如何操作这些组件。执行后的效果如图 3-12 所示。

3. API 应用实例 ApiDemos

ApiDemos 演示了很多 API 的使用方法，包括 app、content、graphic、media 等，具体界面如图 3-13 所示。在图 3-13 中可以选择上面的分类，从而可以选择查看具体的分类，进一步了解 API 的强大功能。

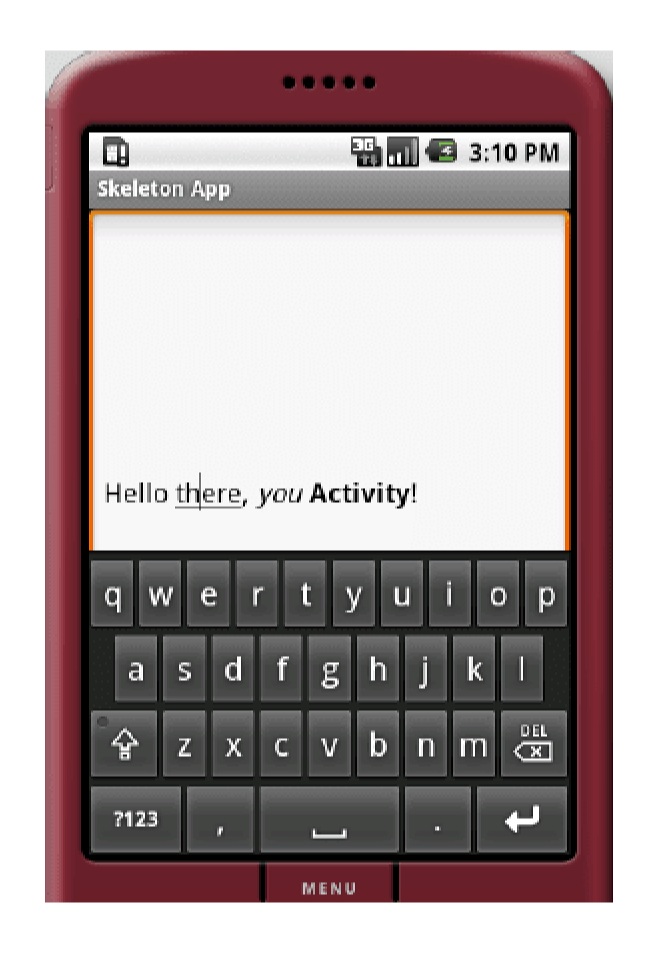


图 3-12　执行效果

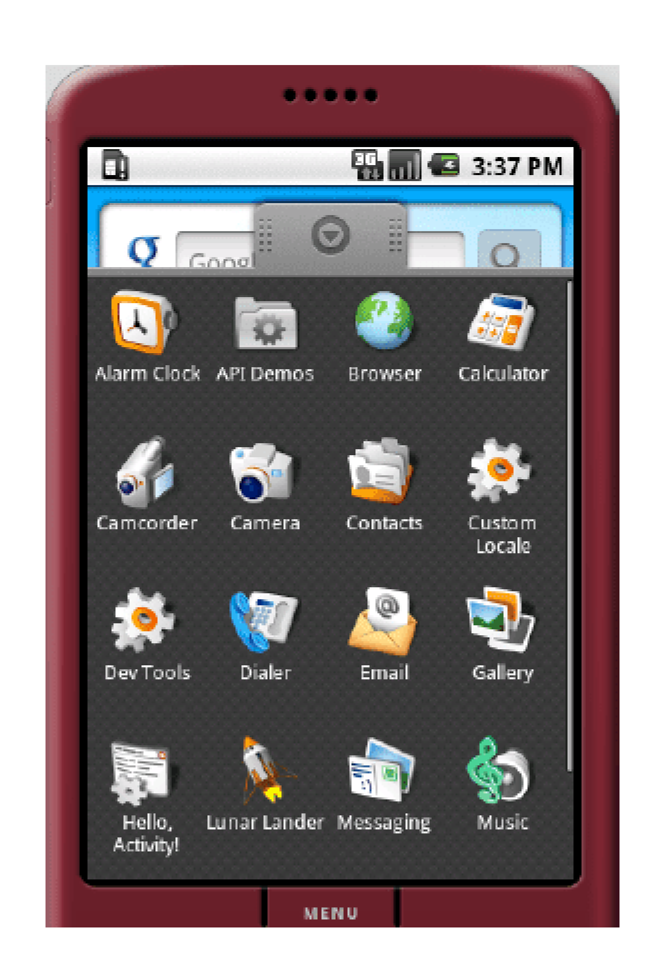


图 3-13　执行效果

4. LunarLander

LunarLander 是一个登月游戏实例，演示了一个类似于登陆月球的小游戏，用户可以通过方向键和点火时机控制画面上的飞船。

5. NotePad

NotePad 是一个记事本程序，此程序可以实现新建、编辑和删除等文档操作。本实例应用了 SQLite 的数据存储和编辑，并使用了 ContentProvider 等方面的信息。

6. Snake

Snake 是贪吃蛇演示实例，这是一款经典的游戏，用户使用手机方向键可以对游戏进行控制。

7. Home

Home 是一款主题类软件实现的实例，实现了一套新的主题界面。此实例演示了如何开发主题类应用，读者通过这个实例可以轻松掌握主题类开发的步骤和一些注意事项。

8. SoftKeyboard

SoftKeyboard 是一个软键盘实例，此实例演示了如何将软键盘绑定到输入框的输入事件上，当焦点到输入框上时，将自动显示软键盘。

9. JetBoy

JetBoy 是一款具备声音支持的游戏实例，它模拟演示了如何在游戏中集成 SONiVOX 的 audioINSIDE 技术，此技术是 SONiVOX 捐赠给手机联盟的。此实例可以完美地播放背景音乐和场景，实现子弹击碎飞来的障碍物等一系列效果。

至此，我终于完成了对 Android SDK 的剖解，通过浏览师兄们的作品，深刻体会到了 Android SDK 的强大功能。我决定要好好修行，也取得和师兄们一样的成就。

注意：读者可以用 Eclipse 亲自调试运行上述实例，查看具体的运行效果。

3.4　第一次考验

师傅：虽然你很快地完成了对 Android SDK 的剖析，但是你莫要高兴太早。今天为师就出一个题目来考考你：利用你手中的武器，实现在手机屏幕中显示问候语“你好！”

我心里想着师傅的问题，看着手里的武器，我知道需要使用 JDK、Eclipse、Android SDK 来实现，在开始之前我做了一个简单的流程规划，具体如图 3-14 所示。

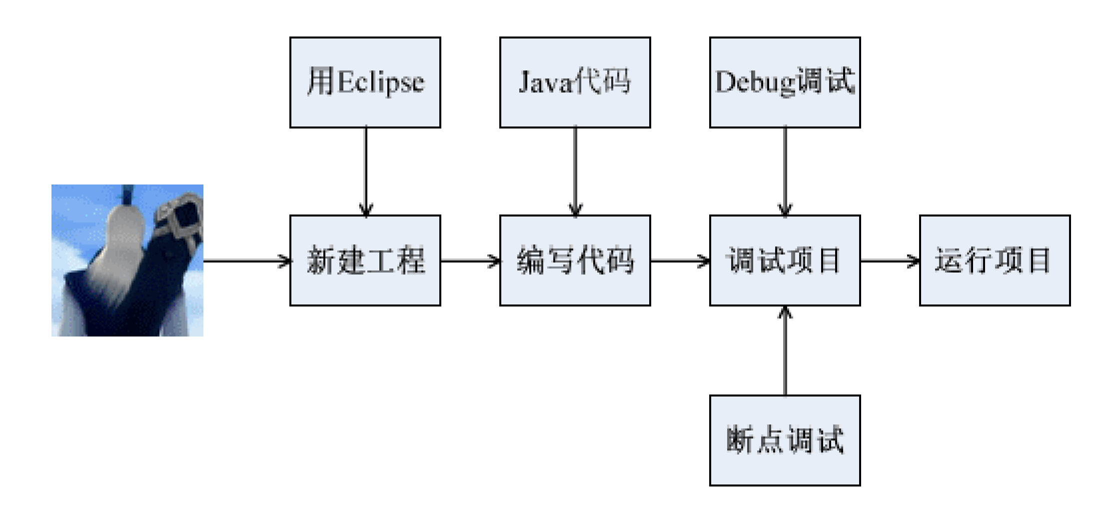


图 3-14　我规划的流程

3.4.1　新建 Android 工程

第 1 步：在 Eclipse 中依次选择 File | New | Project 菜单命令，新建一个工程文件，如图 3-15 所示。

第 2 步：选择 Android Project 选项，单击 Next 按钮。

第 3 步：在弹出的 New Android Project 对话框中设置工程信息，如图 3-16 所示。

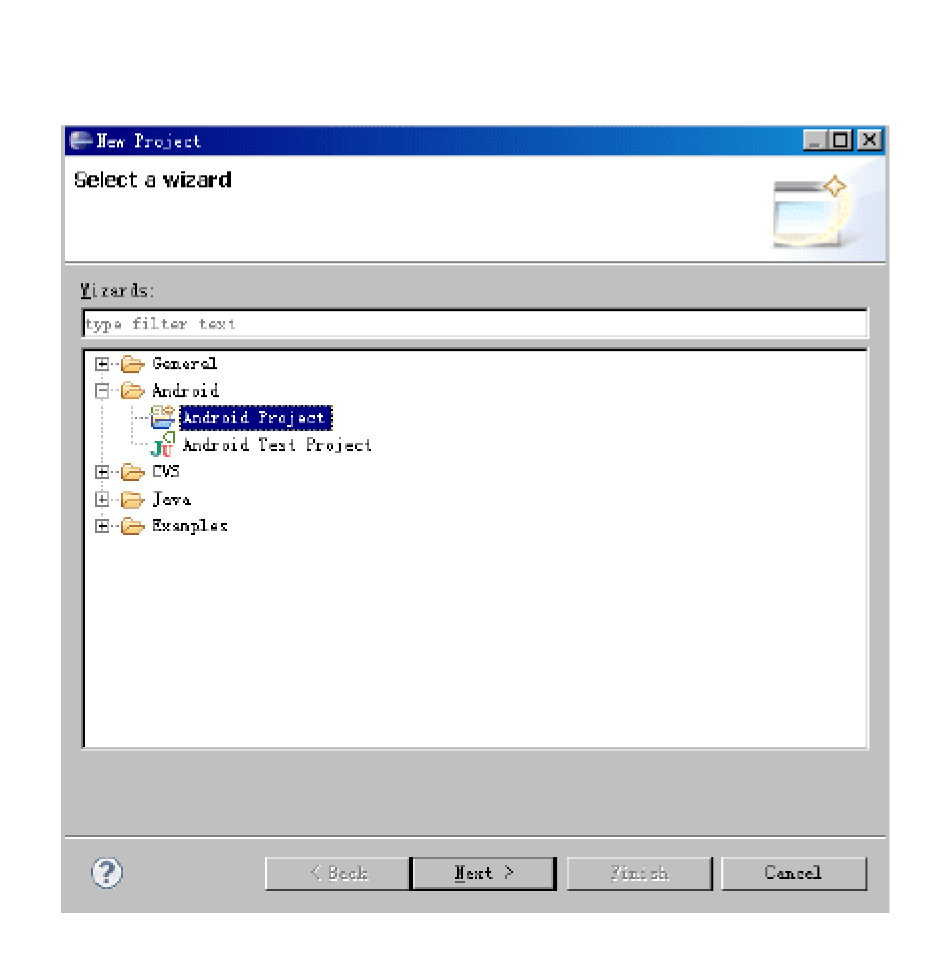


图 3-15　新建工程文件

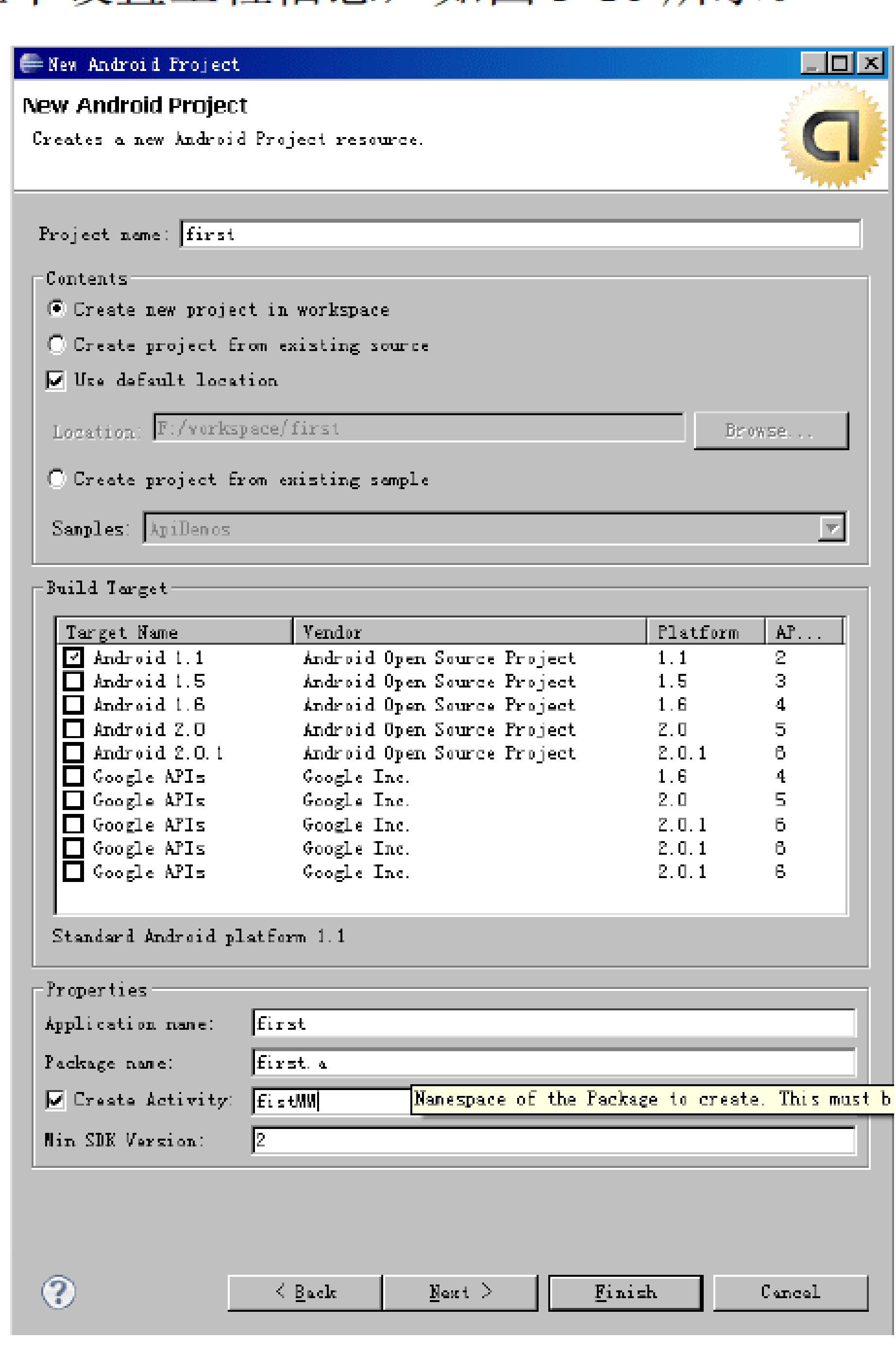


图 3-16　设置工程信息

在图 3-16 所示的对话框中依次设置工程名字、包名字、Activity 名字和应用名字。

3.4.2 编写代码和代码分析

我现在已经创建了一个名为“first”的工程文件，现在打开文件“fistMM.java”，会显示自动生成的如下代码：

```
pacopfirst.a;
import android.app.Activity;
import android.os.Bundle;
public class fistMM extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

如果此时运行程序，将不会显示任何东西。此时我们可以对上述代码进行稍微的修改，让程序输出“你好！”。修改后的具体代码如下所示：

```
package first.a;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class fistMM extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView tv = new TextView(this);
        tv.setText("你好！");
        setContentView(tv);
    }
}
```

经过改写上述代码后，我觉得应该可以在屏幕中输出“你好！”了，可以完全满足师傅题目的要求。

代码编写完毕后，我想整个题目已基本完成了，便忙不迭地向师傅去请功，谁知道师傅给了我当头一棒。

做任何事情都不要太急功近利，特别像你这样的入门弟子，基本功一定要打得扎实。现在你对代码还不熟悉，甚至对开发环境都不熟悉，所以更应该一步一个脚印地进行。现在还不能直接运行程序，而是应该调试程序代码，看看是否有错误。

3.4.3 调试

师傅说 Android 调试一般分成 3 个步骤：设置断点、Debug 调试和断点调试。

1)　设置断点

此处的设置断点和 Java 中的方法一样，可以通过双击代码左边的区域进行断点设置，如图 3-17 所示。为了调试方便，可以设置显示代码的行数，在代码左侧的空白部分单击鼠标右键，然后在弹出的菜单命令中选择 Show Line Numbers 选项，如图 3-17 所示。

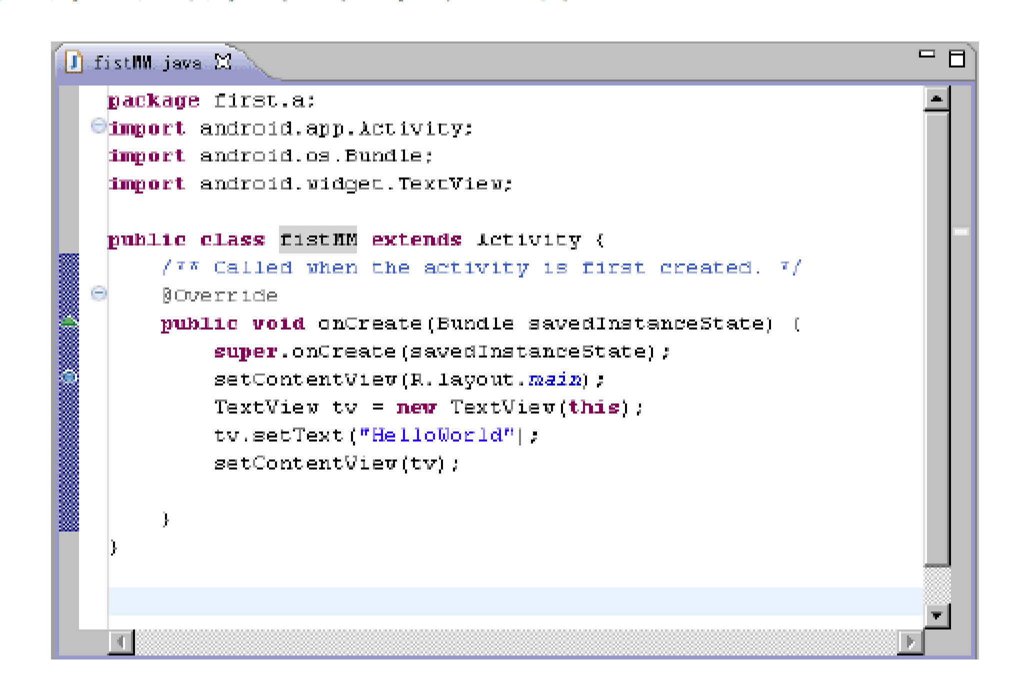


图 3-17　设置断点

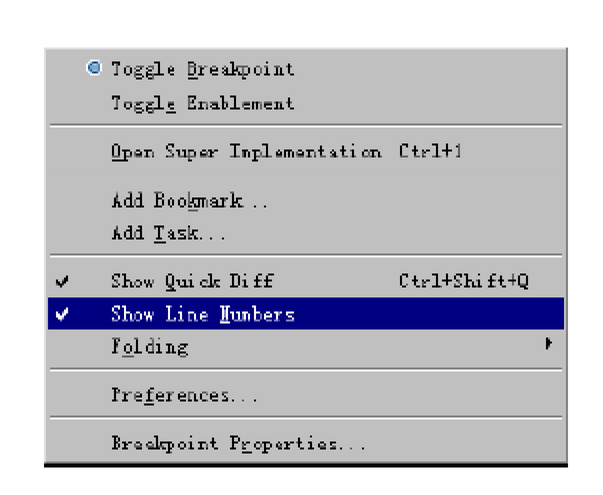


图 3-18　显示行数

2)　Debug 调试

Debug Android 项目操作和普通的 Debug Java 项目类似，只是在选择调试项目时选择 Android Application 即可。即右键单击项目名，在弹出的菜单中依次选择 Debug As | 1 Android Application 命令，如图 3-19 所示。

3)　断点调试

通过设置断点可以进行单步调试，具体调试方法和普通的 Java 程序类似，具体调试界面如图 3-20 所示。

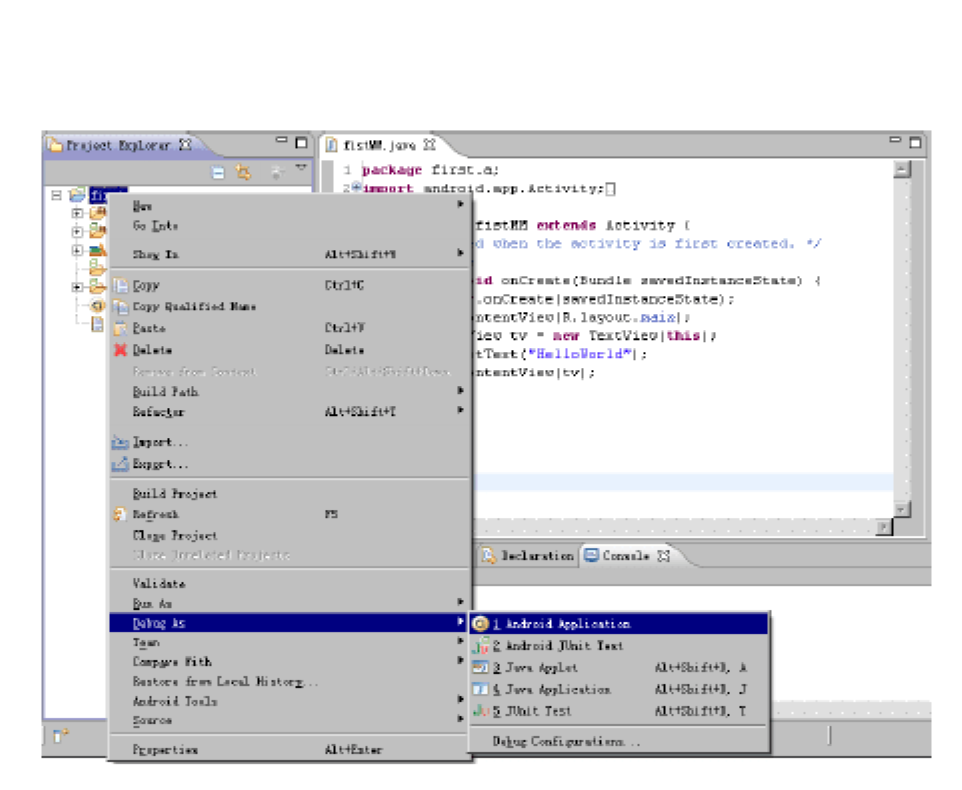


图 3-19　Debug 高度

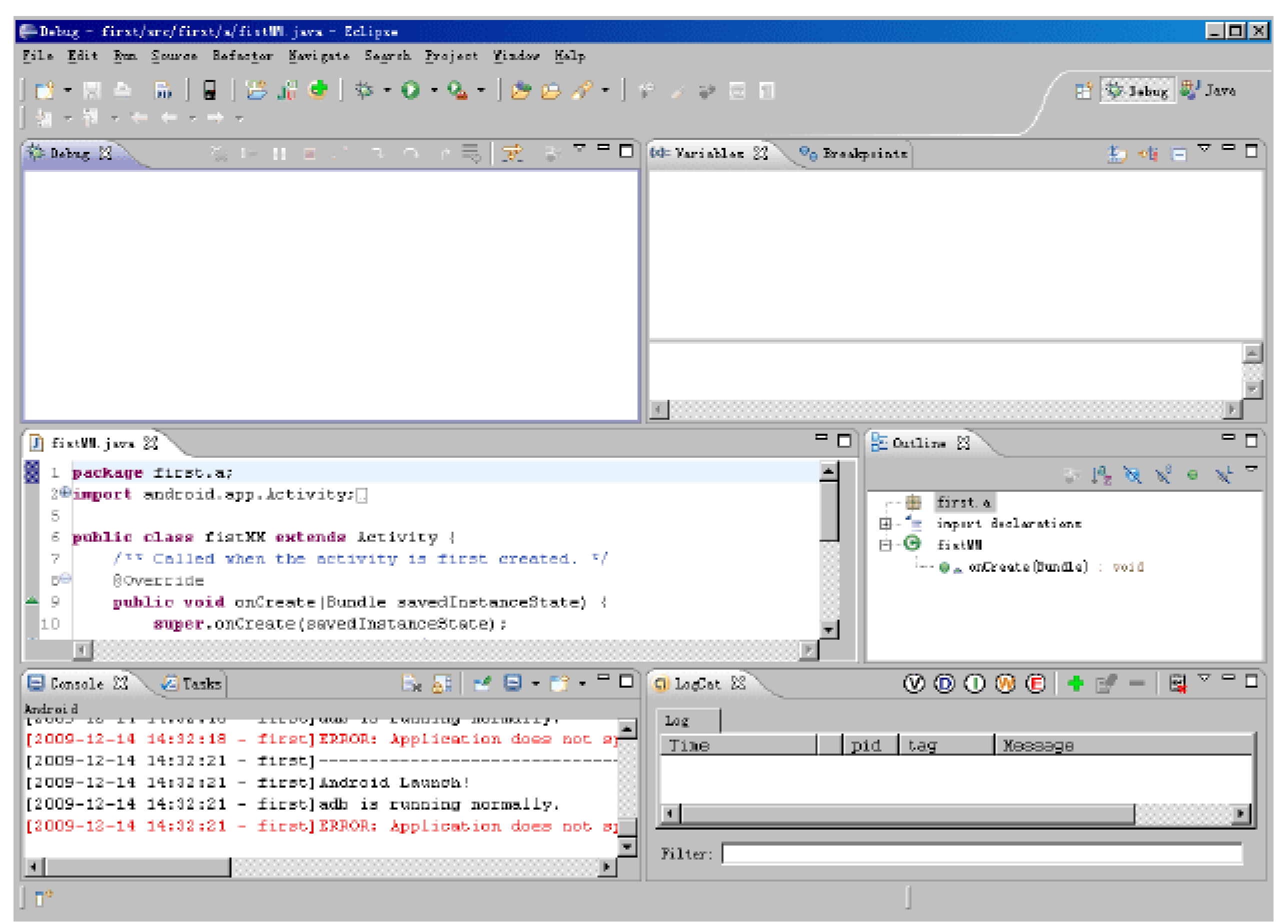


图 3-20　调试界面

3.4.4　运行项目

使用 Eclipse 打开在 3.4.2 中编写的代码，接下来开始运行这段程序，具体过程如下。

第 1 步：右键单击项目名，在弹出的菜单中依次选择 Run As | Android Application 命令，如图 3-21 所示。

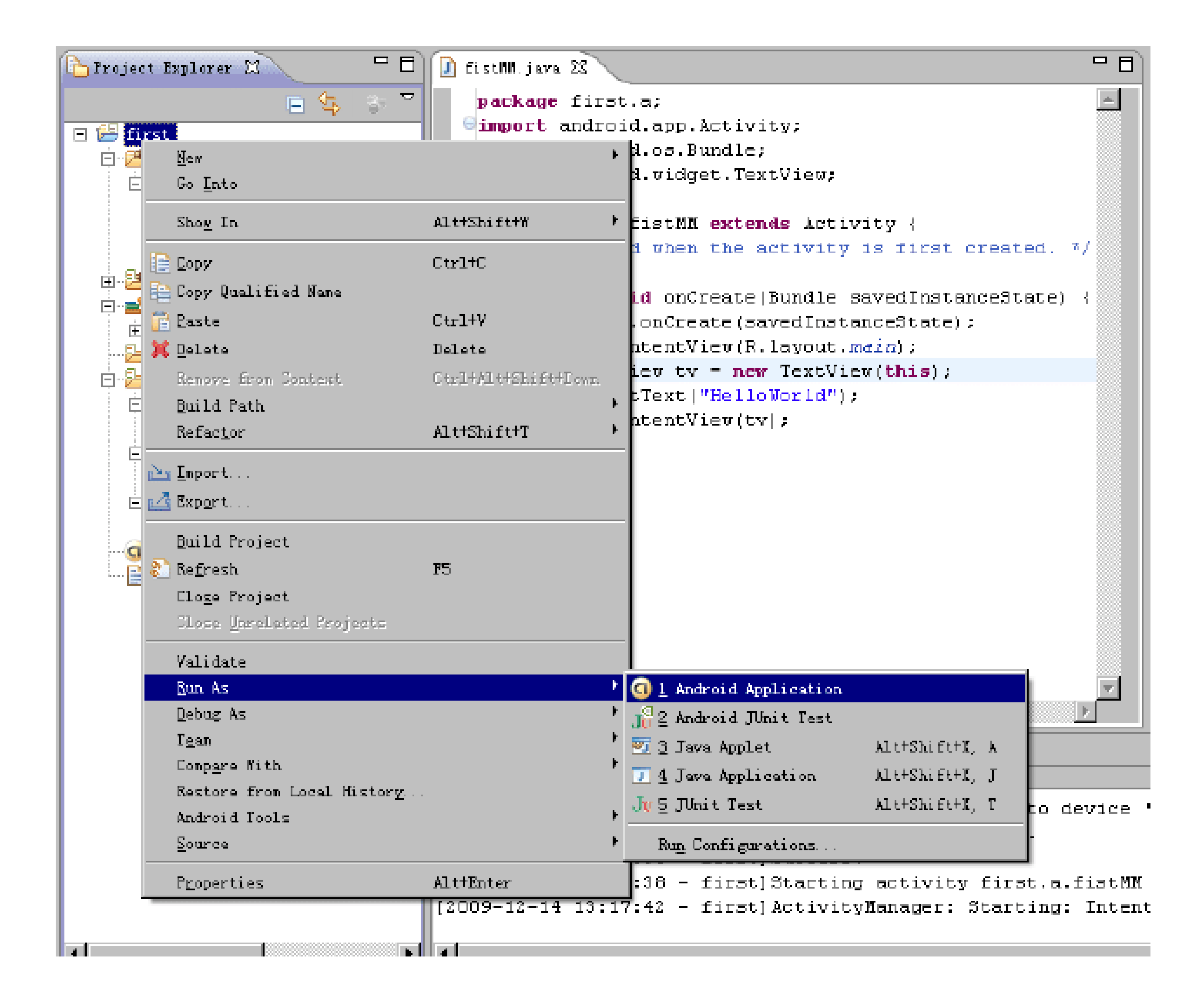


图 3-21　开始调试

第 2 步：工程开始运行，运行完成后在屏幕中输出“你好！”，如图 3-22 所示。

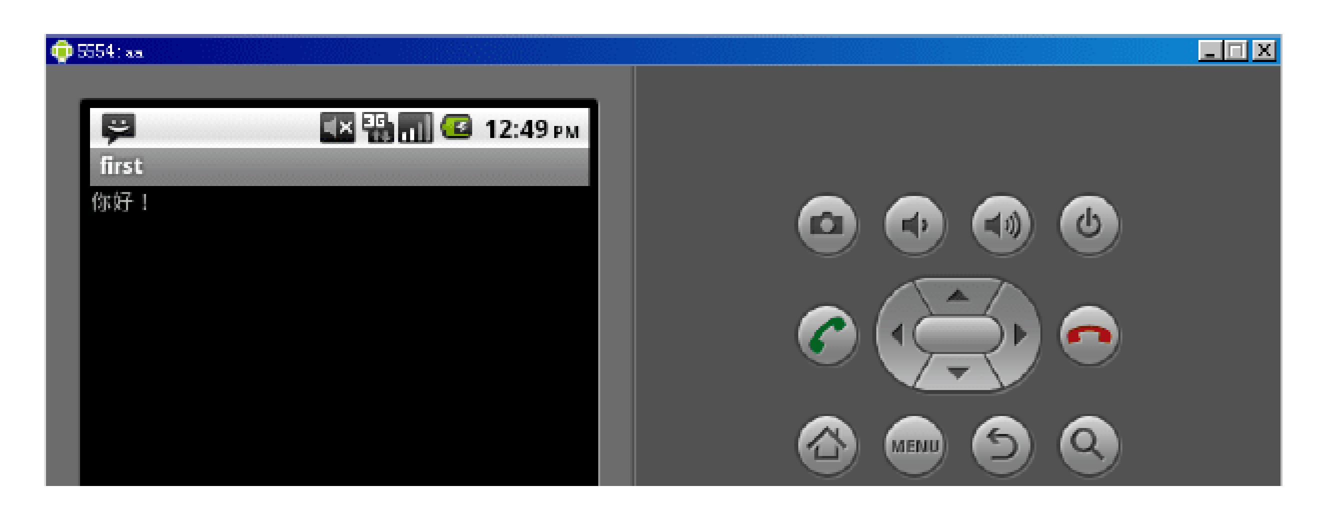


图 3-22　运行结果

提醒：如果加载的时间太长，运行后可能会输出如图 3-22 所示的界面。

解决上述问题的方法是：单击图 3-23 中的 MENU 键，然后在弹出的窗口中单击 Wait 按钮，如图 3-24 所示。

经过上述操作，即可成功显示运行效果。

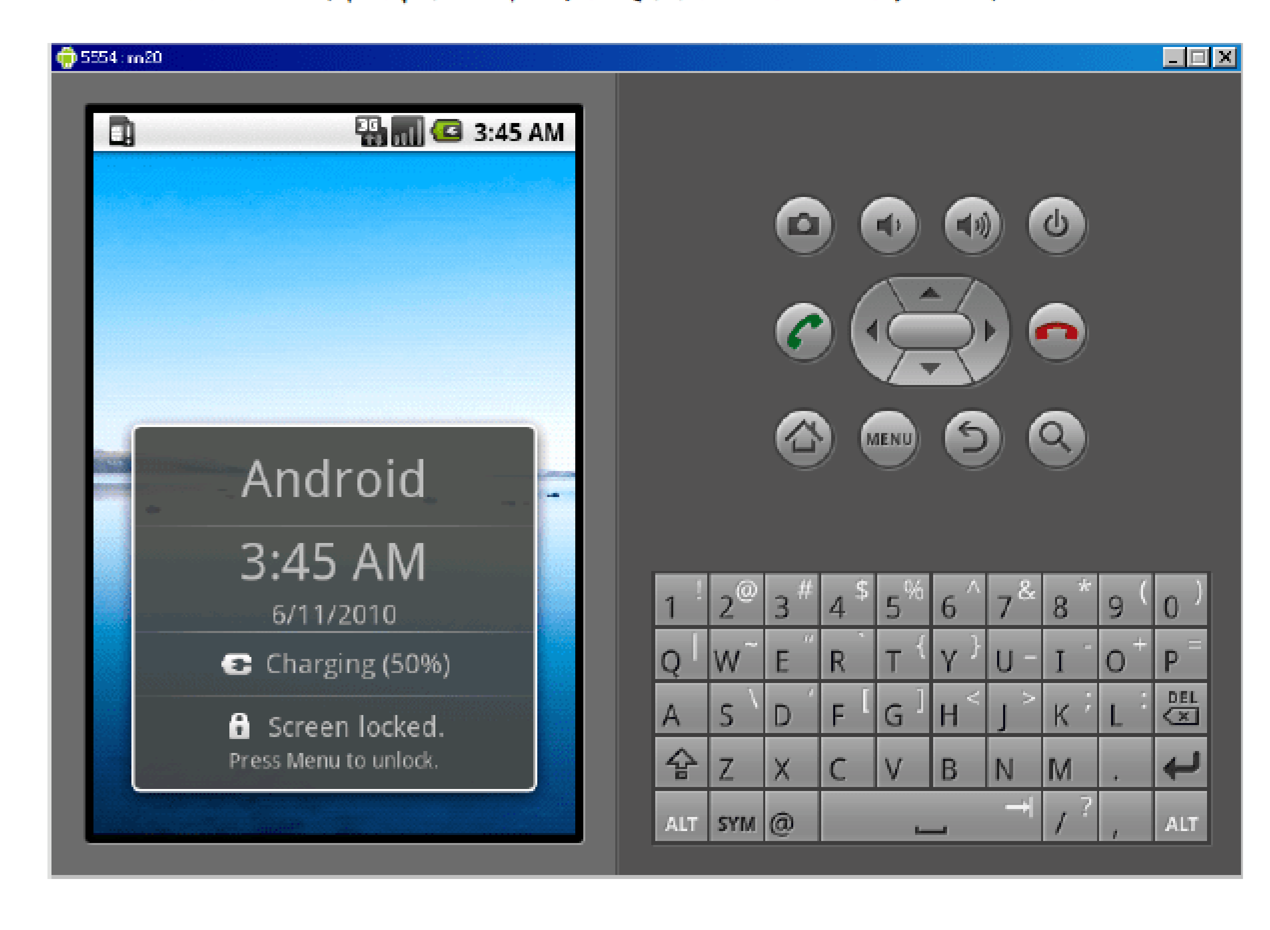


图 3-23　加载时间过长出现的界面



图 3-24　单击 Wait 按钮



第二篇　核心技术篇

第 4 章　勤练心法——Android 应用核心

在前面的章节中，已经讲解了 Android 开发环境的搭建过程和 Android SDK 框架的基本知识，并且通过简单的实例程序，分解了 Android 的各段代码。本章将进一步分解 Android 应用程序，详细剖析 Android 应用程序的核心构成部分，为读者步入后面章节的学习打下基础。

4.1　当 头 一 棒

过去几天的修炼，自我感觉对 Android SDK 的使用已经是得心应手。最近几天我一直憧憬着在江湖中行侠仗义的情景，决定向师傅请示下山闯荡一番，但是师傅下面的一番话打消了我的念头。

不要以为自己的水平已经足够高，因为江湖太过凶险，如果功夫不到家，最后弄得下岗或者被开除的结果就悔之晚矣。练功之路是条充满了挑战的路，必须循序渐进，寻找捷径是不现实的。在此为师给你以下 3 条警告。

1)　培养兴趣

兴趣是能够让我们坚持学习 Android 开发的动力。无论做什么事情，只要有了兴趣，你就喜欢花费时间去做它。只要你喜欢感受那种调试程序成功的喜悦，就说明你已经对编程产生了兴趣。

2)　脚踏实地

欲速则不达，学 Android SDK 开发切忌有浮躁的心态。很多刚入门的弟子只学会了基本语法，调试成功了几段代码，就迫不及待地大声宣布“我精通 Android SDK 开发了”。但是当遇到问题之后才发现，自己学到的只是九牛一毛。

3) 多实践

软件开发很强调动手能力，所以实践就变得尤为重要。本派前辈高人认为，学习 Android SDK 开发的秘诀是：练习，练习，再练习。建议你每听完我传授的武功心法后，不要等到完全理解了才实践，而是应该边默念心法边敲代码，程序运行的各种情况可以让你更快更牢固地掌握知识点。

说完这些，想必你也应该能体会到为师的一片苦心了。你还是继续练功吧，今天为师传授给你“Android 应用核心”心法，这个心法能让你了解我们 Android 派的内部组织结构，也能了解各个内部组织间的运作流程。

4.2 Android 体系结构

Android 作为当前移动平台的新兴门派，其层次结构吸取了其他各大门派之长，严谨、科学。它包括操作系统(OS)、中间件(MiddleWare)和应用程序(Application)。

Android 的整个组织结构自下而上分为以下几个层次。

(1) 操作系统层(OS)。

(2) 各种库(Libraries)和 Android 运行环境(RunTime)。

(3) 应用程序框架(Application Framework)。

(4) 应用程序(Application)。

上述层次的具体结构如图 4-1 所示。

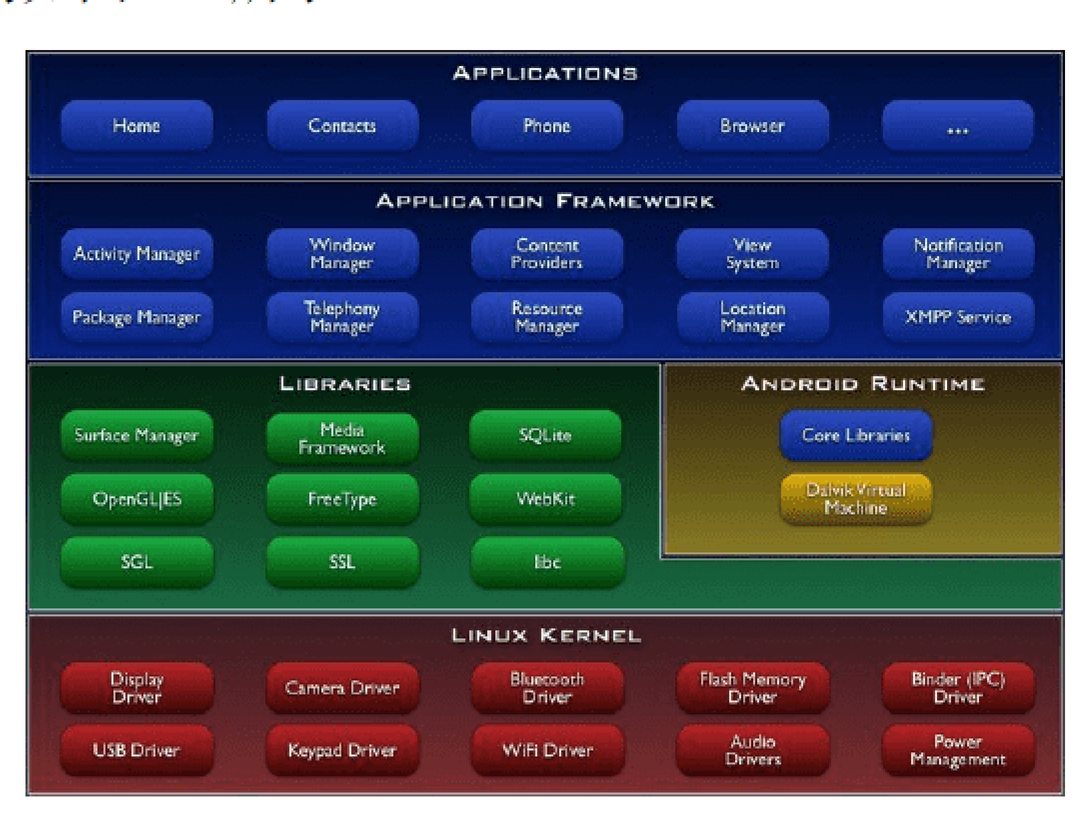


图 4-1 Android 组件结构图

4.2.1 库和 Android 运行环境是根基

这是整个组织内最底层的部分，也是最基本的部分，是安卓派的根基。本层对应的 Linux 世家成员一般是嵌入式系统，相当于中间件层次。本层将分成两个部分：一个是各种库；另一个是运行环境。本层的内容大多是使用 C++实现的，其中包含的各种库如下。

- C 库：C 语言的标准库，也是系统中一个最底层的库，C 库是通过 Linux 的系统调用来实现的。
- 多媒体框架(Media Frameword)：此部分内容是 Android 多媒体的核心部分，基于 PacketVideo(即 PV)的 OpenCORE。本库可以分为两大部分：一部分是音频、视频的回

放(PlayBack)，另一部分是音、视频的记录(Recorder)。

- SGL：2D 图像引擎。
- SSL：即 Secure Socket Layer，位于 TCP/IP 协议与各种应用层协议之间，能够为数据通讯提供安全支持。
- OpenGL ES 1.0：提供了对 3D 的支持。
- 界面管理工具(Surface Management)：提供了管理显示子系统等功能。
- SQLite：是一个通用的嵌入式数据库。
- WebKit：网络浏览器的核心。
- FreeType：位图和矢量字体的功能。

我发现 Android 的各种库一般都是以系统中间件的形式提供的，它们均有一个显著的特点——与移动设备平台的应用密切相关。

4.2.2　应用程序框架是中间层

Android 的应用程序框架(Application Framework)为应用程序层的开发者提供 APIs。由于上层的应用程序是以 Java 构建的，因此本层首先包含了 UI 程序中所需要的各种控件，如 Views(视图组件)，其中又包括了 Lists(列表)、Grids(栅格)、Text Boxes(文本框)、Buttons(按钮)等，甚至包含了一个嵌入式的 Web 浏览器。

一个基本的 Android 应用程序可以利用应用程序框架中的以下几个部分：

- Activity(活动)；
- Broadcast Intent Receiver(广播意图接收者)；
- Service(服务)；
- Content Provider(内容提供者)。

4.2.3　操作系统层是根本

安卓起源于 Linux 世家，用 Linux 2.6 作为操作系统。Linux 2.6 是江湖传统的四大世家之一，是一种标准的操作系统技术，同时也是一个开放的操作系统。Android 对操作系统的使用包括核心和驱动程序两部分，Android 的 Linux 核心是标准的 Linux 2.6 内核，其实 Android 更多应用是与移动设备相关的驱动程序。主要的驱动程序如下所示。

- 显示驱动(Display Driver)：常用基于 Linux 的帧缓冲(Frame Buffer)驱动。
- Flash 内存驱动(Flash Memory Driver)：基于 SD 等闪存设备应用的驱动程序。
- 照相机驱动(Camera Driver)：常用基于 Linux 的 v4l(Video for)驱动。
- 音频驱动(Audio Driver)：常用基于 ALSA(Advanced Linux Sound Architecture，高级 Linux 声音体系)驱动。
- WiFi 驱动(Camera Driver)：基于 IEEE 802.11 标准的驱动程序。
- 键盘驱动(KeyBoard Driver)：基于按键应用的驱动程序。
- 蓝牙驱动(Bluetooth Driver)：基于蓝牙设备的驱动程序。
- Binder IPC 驱动：一个特殊的 Android 驱动程序，具有单独的设备节点，提供进程间的通讯功能。
- Power Management(能源管理)：基于电源显示应用的驱动程序。

4.2.4 应用程序

Android 的应用程序(Application)主要是用户界面(User Interface)方面的，通常用 Java 程序来编写，其中还可以包含各种资源文件(放置在 res 目录中)、Java 程序及相关资源，经过编译后将生成一个 APK 包。Android 本身提供了主屏幕(Home)、联系人(Contact)、电话(Phone)、浏览器(Browser)等众多的核心应用。同时，应用程序的开发者还可以使用应用程序框架层的 API 实现自己的程序，这也是 Android 开源的巨大潜力体现。

4.3 Android 应用程序组成

经过对 Android 体系结构的了解，我知道了整个门派内的各个组织结构，了解了门派的根基和层次成员的构成。都说要学以致用，看来基础和中间层次在现实中直接用到的时候较少，用得比较多的应该是应用程序中的内容。

一个典型的 Android 应用程序通常由 5 个组件组成，具体如图 4-2 所示。

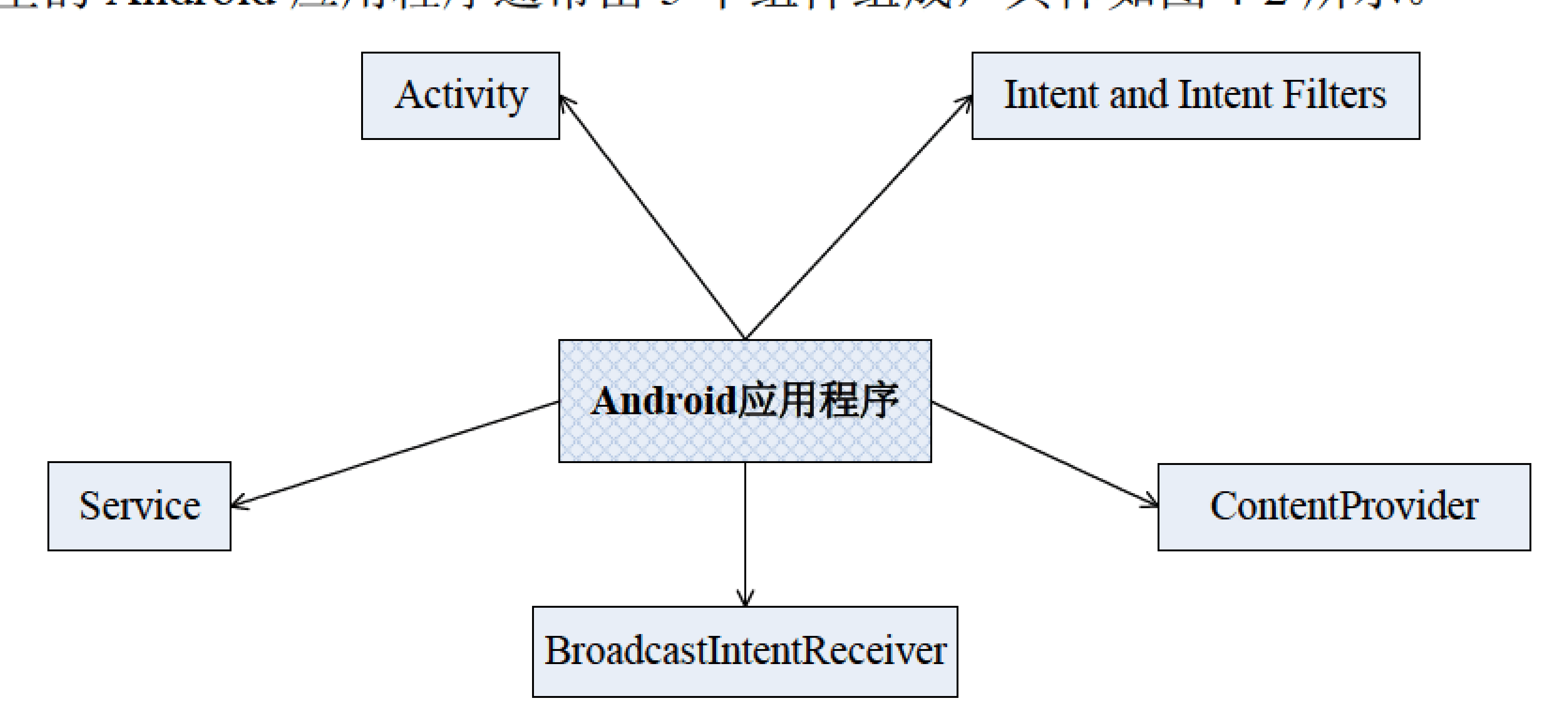


图 4-2 典型 Android 应用程序的 5 个组件

4.3.1 Activity

Activity 是这 5 个组件中最常用的。程序中 Activity 通常的表现形式是一个单独的界面(screen)。每个 Activity 都是一个单独的类，它扩展实现了 Activity 基础类。这个类显示为一个由 Views 组成的用户界面，并响应事件。大多数程序有多个 Activity，例如，一个文本信息程序有这么几个界面：显示联系人列表界面、写信息界面、查看信息界面或者设置界面等。每个界面都是一个 Activity，切换到另一个界面就是载入一个新的 Activity。某些情况下，一个 Activity 可能会给前一个 Activity 返回值，例如，一个让用户选择相片的 Activity 会把选择到的相片返回给其调用者。

打开一个新界面后，前一个界面就被暂停，并放入历史栈中(界面切换历史栈)。使用者可以回溯前面已经打开的存放在历史栈中的界面，也可以从历史栈中删除没有界面价值的界面。Android 在历史栈中保留程序运行产生的所有界面：从第一个界面，到最后一个界面。

4.3.2　Intent and Intent Filters

Android 通过一个专门的 Intent 类来进行界面的切换，Intent 描述了程序想做什么。数据结构的两个最重要的部分是操作(action)与按照既定规则处理数据(data)。典型的操作是 main (Activity 的入口)、view、pick、edit 等。数据用 URI 表示，例如，查看某人的联系信息，你需要创建一个 Intent，使用 view 操作，数据则是一个指向此人的 URI。

有个相关的类叫 IntentFilter。Intent 请求做什么事情；IntentFilter 则描述了一个 Activity(或下文的 IntentReceiver)能处理什么意图。显示某人联系信息的 Activity 使用了一个 IntentFilter，就是说它知道如何处理应用到此人数据的 view 操作。Activity 在 AndroidManifest.xml 文件中使用 IntentFilters。

通过解析 Intents 来完成 Activity 的切换，使用 startActivity(myIntent)来启用新的 Activity。系统考察所有安装程序的 IntentFilters，然后找到与 myIntent 匹配最好的 IntentFilters 所对应的 Activity。这个新 Activity 接到 Intent 传来的消息，并因此被启用。解析 Intents 的过程发生在 startActivity 被实时调用时，这样做有以下两个好处。

(1)　Activity 仅发出一个 Intent 请求，便能重用其他组件的功能。

(2)　Activity 可以随时被替换为有等价 IntentFilter 的新 Activity。

4.3.3　Service 介绍

Service 是一个没有 UI 且长驻系统的代码。最常见使用 Service 的例子是媒体播放器从播放列表中播放歌曲这一应用。媒体播放器程序中，可能有一个或多个 Activity 让用户选择歌曲播放。然而，在后台播放歌曲就无需 Activity 干涉了，因为用户希望在音乐播放的同时能够切换到其他界面。这样，媒体播放器 Activity 需要通过 Context.startService()启动一个 Service，这个 Service 在后台运行以保持继续播放音乐。在媒体播放器被关闭之前，系统会保持音乐后台播放 Service 的正常运行。可以用 Context.bindService()方法连接到一个 Service 上(如果 Service 未运行的话，连接后还会启动它)。连接上之后就可以通过一个 Service 提供的接口与 Service 进行通话。对音乐 Service 来说，则提供了暂停、重放等功能。

1. 如何使用服务

使用服务有以下两种方法。

(1)　通过调用 Context.startService()启动，调用 Context.stopService()结束，startService()可以传递参数给 Service。

(2)　通过调用 Context.bindService()启动，调用 Context.unbindService()结束，还可以通过 ServiceConnection 访问 Service。二者可以混合使用，比如说我可以先用 startService()启动服务，再用 unbindService()结束服务。

2. Service 的生命周期

在使用 startService()启动服务后，即使调用 startService()的进程结束了，Service 进程仍然还存在，直到有进程调用 stopService()，或者 Service 自己灭亡(stopSelf())为止。

在使用 bindService()绑定服务后，Service 就和调用 bindService()的进程同生共死，也就是说

当调用 bindService()的进程死了，那么它绑定的 Service 也要跟着被结束，当然期间也可以调用 unbindService()让 Service 进程结束。

那么只有你用 StopService()停止了服务，而且我用 unbindService()解除了服务，这个服务进程才会被结束。

3. 进程生命周期

Android 系统将会尝试保留那些启动了的或者是绑定了的服务进程，具体说明如下。

- 如果该服务正在进程的 onCreate()、onStart()或者 onDestroy()这些方法中执行，那么主进程将会成为一个前台进程，以确保此代码不会被停止。
- 如果服务已经开始，那么就重要性而言它的主进程会低于所有可见的进程但高于不可见的进程，由于只有少数几个进程是用户可见的，所以只要不是内存特别低，该服务是不会停止的。
- 如果有多个客户端绑定了服务，那么只要客户端中的一个对用户是可见的，即认为该服务可见。

4.3.4 BroadcastIntentReceiver

当要执行一些与外部事件相关的代码时，就可能需要使用 IntentReceiver 了。IntentReceivers 没有 UI，尽管它们使用 NotificationManager 来通知用户一些好玩的事情发生了。IntentReceivers 在 AndroidManifest.xml 文件中声明，不过用户可以使用 Context.registerReceiver()来声明。你的程序没有必要运行声明来等待 IntentReceivers 被调用。当一个 IntentReceiver 被触发时，如何需要的话，系统自然会启动你的程序。程序也可以通过 Context.broadcastIntent()来发送自己的 Intent 广播给其他程序。

4.3.5 ContentProvider

应用程序把数据存放在一个 SQLite 数据库格式的文件里，或者存放在其他有效设备里。如果你想让其他程序能够使用你自己程序的数据，ContentProvider 就很有用了。ContentProvider 是一个实现了一系列标准方法的类，这个类使得其他程序能存储、读取某种 ContentProvider 可处理的数据。

4.4 Android 应用工程文件组成

我对整个 Android 体系结构和典型应用程序的内容了解得差不多了。接下来听从师傅的安排，开始仔细分析 Android 应用工程文件。我发现 Android 应用工程文件主要由以下工程文件组成。

- src 文件：源文件都在这个目录里面。
- R.java 文件：这个文件是 Eclipse 自动生成的，应用开发者不需要去修改里边的内容。
- Android.jan：这个是应用运行的 Android 库。
- assets 目录：里面主要放置多媒体等一些文件。

- res 目录：里面主要放置应用的资源文件。
- values 目录：主要放置字符串(strings.xml)、颜色(colors.xml)和数组(arrays.xml)。
- Androidmanifest.xml：相当于应用的配置文件。在这个文件中，必须声明应用的名称，应用所用到的 Activity、Service 以及 receiver 等。
- drawable 目录：主要放置应用到的图片资源。
- layout 目录：主要放置用到的布局文件。这些布局文件都是 xml 文件。

在 Eclipse 中，一个基本的 Android 项目的目录结构如图 4-3 所示。

图 4-3　Android 应用工程文件组成

4.4.1　AndroidManifest.xml 文件

AndroidManifest.xml 文件仿佛是一个桌子。在里面包含了项目中所使用的 Activity、Service 和 Receiver。如果打开 HelloAndroid 项目中的 AndroidManifest.xml 文件，会显示类似下面的代码：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="first.a"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".fistMM"
               android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
   <uses-sdk android:minSdkVersion="2" />
</manifest>
```

在上述代码中，intent-filters 描述了 Activity 启动的位置和时间。每当一个 Activity(或者操作系统)要执行一个操作时，它将创建出一个 Intent 对象，这个 Intent 对象能承载的信息可描述你想做什么，你想处理什么数据，数据的类型以及一些其他信息。而 Android 则会和每个 Application 所暴露的 intent-filter 的数据进行比较，找到最合适的 Activity 来处理调用者所指定的数据和操作。也就是说，我们究竟吃哪一桌的菜，是由 intent-filter 所决定的。

接下来，我决定仔细分析 AndroidManifest.xml 文件，具体如表 4-1 所示。

表 4-1　AndroidManifest.xml 这种分析

参　数	说　明
manifest	根节点，包含了 package 中所有的内容
xmlns:android	包含命名空间的声明。xmlns:android=http://schemas.android.com/apk/res/android，使得 Android 中各种标准属性能在文件中使用，提供了大部分元素中的数据
Package	声明应用程序包
application	包含了 package 中 application 级别组件声明的根节点。此元素也可包含 application 的一些全局和默认的属性，如标签、icon、主题、必要的权限等。一个 manifest 能包含零个或一个此元素(不能大于一个)
android:icon	应用程序图标
android:label	应用程序名字
Activity	与用户交互的主要工具。Activity 是用户打开一个应用程序的初始页面，大部分被使用到的其他页面也由不同的 Activity 来实现，并声明在另外的 Activity 标记中。注意，每一个 Activity 必须有一个<activity>标记对应，无论它给外部使用还是只用于自己的 package 中。如果一个 Activity 没有对应的标记，你将不能运行它。另外，为了支持运行时查找 Activity，可包含一个或多个<intent-filter>元素来描述 activity 所支持的操作
android:name	应用程序默认启动的 Activity
intent-filter	声明了指定的一组组件支持的 Intent 值，从而形成了 Intent-filter。除了能在此元素下指定不同类型的值，属性也能放在这里，用来描述一个操作所需的唯一标签、icon 和其他信息
action	组件支持的 Intent action
category	组件支持的 Intent category，这里指定了应用程序默认启动的 Activity
uses-sdk	该应用程序所使用的 SDK 版本

4.4.2　src 目录

和传统的 Java 项目一样，安卓工程的 src 文件夹是项目的所有包及源文件(.java)，而在 res 文件夹中则包含了项目中的资源，例如常见的程序图标(drawable)、布局文件(layout)、值(values)等。但是安卓毕竟是安卓，也有 Java 中没有的东西。相对于 Java，安卓中所特有的是 gen 文件夹中的 R.java 文件和每个 Android 项目都必须有的 AndroidManifest.xml 文件。

在 src 目录中，.java 格式文件是在建立项目时自动生成的，这个文件是只读模式，不能更改。R.java 文件是定义该项目所有资源的索引文件。下面看看前面 HelloAndroid 项目中 R.java 文件的代码：

```
package com.yarin.Android.HelloAndroid;
public final class R {
   public static final class attr {
   }
   public static final class drawable {
      public static final int icon=0x7f020000;
   }
```

```
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

在上述代码中，我看到了很多定义的常量，并且这些常量的名字都与 res 文件夹中的文件名相同，这说明.java 文件中所存储的是该项目所有资源的索引。有了这个文件后，在程序中使用资源将变得更加方便，可以很快地找到要使用的资源，由于这个文件不能被手动编辑，所以当我们在项目中加入了新的资源时，只需要刷新一下该项目，.java 文件便自动生成了所有资源的索引。

4.4.3　值的定义文件

在安卓项目中，由一个专用的文件来定义常量值。例如，下面是资源文件中常量的定义文件 String.xml，具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Hello Android!</string>
    <string name="app name">Hello Android</string>
</resources>
```

上述代码十分简单，只是简单定义了两个字符串资源。

接下来开始分析 Hello Android 项目的布局文件(layout)，在前面项目中的主布局文件是 main.xml，其对应代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout width="fill parent"
    android:layout height="wrap content"
    android:text="@string/hello"
    />
</LinearLayout>
```

在上述代码中有以下几个布局和参数。

- ❑ < LinearLayout>：线性版面配置，在这个标签中，所有元件都是按由上到下的顺序排成的。
- ❑ android:orientation="vertical"：表示这个介质的版面配置方式是从上到下垂直地排列其内部的视图。
- ❑ android:layout_width：定义当前视图在屏幕上所占的宽度，fill_parent 即填充整个屏幕。

- android:layout_height：定义当前视图在屏幕上所占的高度，fill_parent 即填充整个屏幕。
- wrap_content：随着文字栏位的不同而改变这个视图的宽度或高度。
- layout_weight：用于给一个线性布局中的多个视图的重要度赋值。所有视图都有 layout_weight 值，默认为零，即需要显示多大的视图就占据多大的屏幕空间。如果值大于零，则将父视图中的可用空间分割，分割大小取决于每一个视图的 layout_weight 值和该值在当前屏幕布局的整体 layout_weight 值，以及在其他视图屏幕布局的 layout_weight 值中所占的比例。

在这里，布局中设置了一个 TextView，用来配置文本标签 Widget，其中设置的属性 android:layout_width 为整个屏幕的宽度，android:layout_height 可以根据文字来改变高度，而 android:text 则设置了这个 TextView 要显示的文字内容，这里引用了@string 中的 hello 字符串，即 String.xml 文件中的 hello 所代表的字符串资源。hello 字符串的内容为："Hello World, Hello Android!"，这就是我们在 Hello Android 项目运行时看到的字符串。

注意：上面介绍的文件只是主要文件，在项目中需要我们自行编写。项目中还有很多其他的文件，那些文件也需要我们编写，在此就不一一进行讲解了。

4.5　不是六道轮回的生命周期

在佛教中有"六道轮回"这一说法，他们认为一切有生命的东西，如不寻求"解脱"，就永远在"六道"(天、人、阿修罗、畜生、饿鬼、地狱)中生死相续，没有止息。虽然 Android 程序无需经过这么多轮回，但是 Android 程序也有自己的具体存活时间，这个具体时间就是一个 Android 程序的周期，如果超过这个期限这段程序就没有作用了。

4.5.1　Android 生命周期

Android 是基于 Linux 世家的，所以是构建在 Linux 之上的开源移动开发平台。在 Android 中，多数情况下每个程序都是在各自独立的 Linux 进程中运行。当一个程序或其某些部分被请求时，它的进程就"出生"了；当这个程序没有必要再运行下去且系统需要回收这个进程的内存用于其他程序时，这个进程就"死亡"了。由此可以看出，Android 程序的生命周期是由系统控制而非程序自身直接控制。

师傅：Android 程序的生命周期是由系统控制的，而不是程序自身直接控制。这和桌面应用程序的思维有一些区别，一个桌面应用程序的进程也是在其他进程或用户请求时被创建的，但是往往是在程序自身收到关闭请求后执行一个特定的动作(比如从 main 函数中 return)而导致进程结束。作为一名刚入行的弟子，必须理解不同的应用程序组件(尤其是 Activity、Service 和 Intent Receiver)是如何影响应用程序生命周期的。如果不正确地使用这些组件，可能会导致系统终止正在执行的重要任务的应用程序进程。

4.5.2　Android 进程

最常见的进程生命周期 Bug 的例子是 Intent Receiver(意图接收器)，当 Intent Receiver 在

onReceive()方法中接收到一个 Intent 时，它会启动一个线程，然后返回。一旦返回，系统将认为 Intent Receiver 不再处于活动状态，因而 Intent Receiver 所在的进程也就不再有用了(除非该进程中还有其他的组件处于活动状态)。因此，系统可能会在任意时刻终止该进程以回收占有的内存。这样进程中创建出的那个线程也将被终止。我感觉这样反而会不好，正在迷茫之时，师傅告诉了我解决方法。

师傅：解决这个问题的方法是在 Intent Receiver 中启动一个服务，让系统知道进程中还有处于活动状态的工作。为了使系统能够正确决定在内存不足时应该终止哪个进程，Android 根据每个进程中运行的组件及组件的状态把进程放入一个重要性分级(Importance Hierarchy)中。

进程的类型有很多种，按重要程度排序划分包括以下几种。

1. 前台进程

前台进程(Foreground)与用户当前正在做的事情密切相关。不同的应用程序组件能够通过不同的方法将它的宿主进程移到前台。在如下的任何一个条件下：进程正在屏幕的最前端运行一个与用户交互的活动(Activity)，它的 onResume()方法被调用；或进程有一个正在运行的 Intent Receiver(它的 IntentReceiver.onReceive()方法正在执行)；或进程有一个服务(Service)，并且在服务的某个回调函数(Service.onCreate()、Service.onStart()或 Service.onDestroy())内有正在执行的代码，系统都将把进程移动到前台。

2. 可见进程

可见进程(Visible)有一个可以被用户从屏幕上看到的活动，但不在前台(它的 onPause()方法被调用)。例如，如果前台的活动是一个对话框，以前的活动就隐藏在对话框之后，就会出现这种进程。可见进程非常重要，一般不允许被终止，除非是为了保证前台进程的运行而不得不终止它。

3. 服务进程

服务进程(Service)拥有一个已经用 startService()方法启动的服务。虽然用户无法直接看到这些进程，但它们做的事情却是用户所关心的(如后台 MP3 回放或后台网络数据的上传、下载)。因此，系统将一直运行这些进程，除非内存不足以维持所有的前台进程和可见进程。

4. 后台进程

后台进程(Background)拥有一个当前用户看不到的活动(它的 onStop()方法被调用)。这些进程对用户体验没有直接的影响。如果它们正确执行了活动生命周期，系统可以在任意时刻终止该进程以回收内存，并提供给前面三种类型的进程使用。系统中通常有很多这样的进程在运行，因此，要将这些进程保存在 LRU 列表中，以确保当内存不足时用户最近看到的进程最后一个被终止。

5. 空进程

空进程(Empty)不拥有任何活动的应用程序组件的进程。保留这种进程的唯一原因是在下次应用程序的某个组件需要运行时，不需要重新创建进程，这样可以提高启动速度。

系统将以进程中当前处于活动状态组件的重要程度为基础对进程进行分类。进程的优先级

可能也会根据该进程与其他进程的依赖关系而增长。例如，如果进程 A 通过在进程 B 中设置 Context.BIND_AUTO_CREATE 标记或使用 ContentProvider 被绑定到一个服务(Service)，那么进程 B 在分类时至少要被看成与进程 A 同等重要。

4.5.3 Activity 生命周期

Android 中进程的生命周期大多数时候是由系统管理的，但是由于手机应用的一些特殊性，需要我们更多地关注各个 Android Component 运行时的生命周期模型。在 Android Component 周期模型中说明了手机应用的特殊性，所谓手机应用的特殊性主要是指如下两点。

(1) 手机应用中，在大多数情况下只能在手机上看到一个程序的一个界面。用户除了通过程序界面上的功能按钮在不同的窗体间切换外，还可以通过 Back 键和 Home 键来返回上一个窗口，而用户使用 Back 键或者 Home 键的时机是非常不确定的，任何时候用户都可以使用 Home 键或 Back 键来强行切换当前的界面。

(2) 通常手机上一些特殊事件的发生也会强制地改变当前用户所处的操作状态，例如，无论在任何状态下，当手机来电时，系统都会优先显示电话接听界面。

了解 Component 的生命周期模型有以下两个好处：

(1) 让我们对软件在手机中的运行情况做到心中有数；

(2) 对于程序开发来说，生命周期中的每一个关键事件都会有我们可以覆写(Override)于各种 Component 对应基类型的事件处理方法，了解各种 Component 的生命周期就是让我们在开发程序时，能够明白该怎样去编写各种事件的处理代码。

1. 无瑕疵的 Activity 状态

当 Activity 被创建或销毁时，它们进入或退出 Activity 栈。当它们做这些动作时，它们可能会在如下 4 种状态间进行迁移。

1) Active

当 Activity 在栈的顶端时，它是可见的，有焦点的前台 Activity，用来响应用户的输入。Android 会不惜一切代价来尝试保证它的活跃性，需要的话它会消灭栈中更靠下的 Activity，以保证 Active Activity 需要的资源。当另一个 Activity 变成 Active 状态时，这个就会变成 Paused。

2) Paused

在一些情况下，你的 Activity 可见但不拥有焦点，在这个时刻它就是暂停的。当最前面的 Activity 是全透明或非全屏的 Activity 时，下面的 Activity 就会到达这个状态。当暂停时，这个 Activity 还是被看做是 Active 的，但不接受用户的输入事件。在极端的情况下，Android 会杀死一个 Paused 的 Activity 来恢复资源给 Active Activity。当一个 Activity 完全不可见时，它就变成 Stopped。

3) Stopped

当一个 Activity 不可见，它就“停止”了。这个 Activity 仍然留在内存里，用来保存所有的状态和成员信息；但是，当系统在什么地方需要内存时，它就像“罪犯”一样被拉出去枪毙了。当一个 Activity 停止时，保存数据和当前 UI 状态是很重要的，一旦 Activity 退出或关闭，它就变成 Inactive。

4)　Inactive

当一个曾经被启动过的 Activity 被“杀死”时，它就变成了 Inactive。Inactive Activity 会从 Activity 栈中移除，当它重新显示和使用时需要再次启动。Activity 状态转换过程如图 4-4 所示。

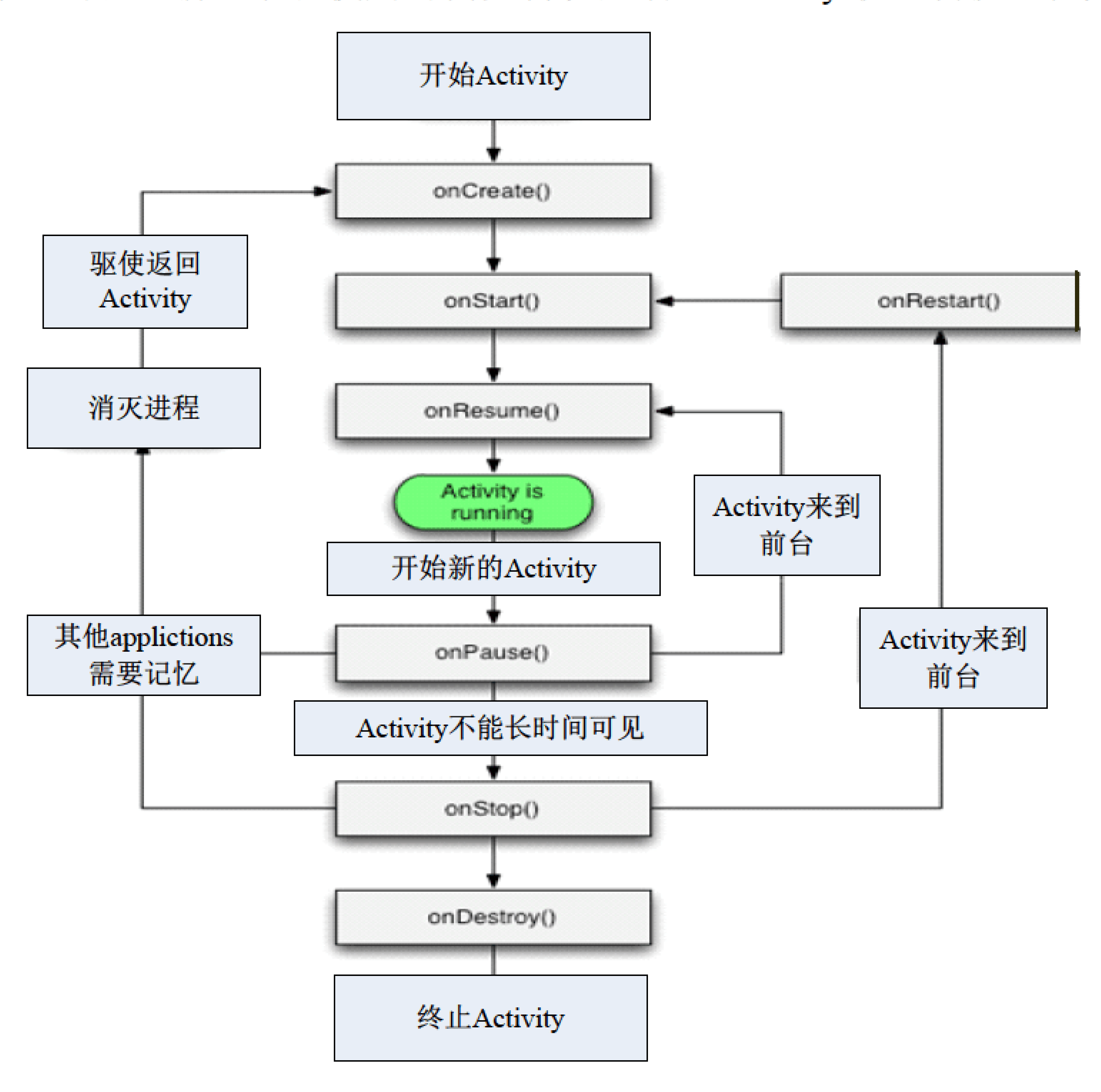


图 4-4　Activity 状态转换图

庆幸的是状态的变化是人为的，完全可以由 Android 内存管理器掌握。Android 会首先关闭那些包含 Inactive Activity 的应用程序，其次关闭那些 Stopped 的程序，极端的情况会移除那些 Paused 的程序。

为了保证无瑕疵的用户体验，这些状态的迁移对用户来说必须是不可见的。当 Activity 从 Paused、Stopped 或者 Inactive 的状态返回到 Active 的时候，UI 必须是无差别的。所以，当 Activity 暂停或停止时，保存所有的 UI 状态和数据是很重要的。一旦 Activity 变成 Active，它需要从保存的数值中恢复。

2. 剖析 Activity

1)　void onCreate(Bundle savedInstanceState)

onCreate 事件在 Activity 被第一次加载时执行。我们新启动一个程序的时候其主窗体的 onCreate 事件就会被执行。如果 Activity 被销毁后(onDestroy 后)，再重新加载进程 Task 时，其 onCreate 事件也会被重新执行。注意这里的参数 savedInstanceState(Bundle 类型是一个键值对集合，大家可以看成是.Net 中的 Dictionary)是一个很有用的设计，由于前面已经说到的手机应用的特殊性，一个 Activity 很可能被强制交换到后台(交换到后台就是指该窗体不再对用户可见，但实际上还是存在于某个 Task 中的。例如，一个新的 Activity 压入了当前的 Task，从而“遮盖”住了当前的 Activity；或者用户按了 Home 键回到桌面；又或者其他重要事件发生，导致新的

Activity 出现在当前的 Activity 之上，比如来电界面)，而如果此后用户在一段时间内没有重新查看该窗体(Android 通过长按 Home 键可以选择最近运行的 6 个程序，或者用户直接再次单击程序的运行图标，如果窗体所在的 Task 和进程没有被系统销毁，则不用重新加载 Process、Task 和 Task 中的 Activity，直接重新显示 Task 顶部的 Activity，这就称之为重新查看某个程序的窗体)，该窗体连同其所在的 Task 和 Process 则可能已经被系统自动销毁了，此时如果再次查看该窗体，则要重新执行 onCreate 事件初始化窗体。而这个时候我们可能希望用户继续上次打开该窗体时的操作状态进行操作，而不是一切从头开始。例如用户在编辑短信时突然来电，接完电话后用户又去做了一些其他的事情，比如保存来电号码到联系人，而没有立即回到短信编辑界面，导致短信编辑界面被销毁，当用户重新进入短信程序时他可能希望继续上次的编辑。这种情况我们就可以覆写 Activity 的 void onSaveInstanceState(Bundle outState)事件，通过向 outState 中写入一些需要在窗体销毁前保存的状态或信息，这样在窗体重新执行 onCreate 的时候，则会通过 savedInstanceState 将之前保存的信息传递进来，此时我们就可以有选择地利用这些信息来初始化窗体。

2) void onStart()

onStart 事件在 onCreate 事件之后执行。或者当前窗体被交换到后台后，在用户重新查看窗体前已经过去了一段时间，窗体已经执行了 onStop 事件，但是窗体和其所在的进程并没有被销毁，用户再次重新查看窗体时会执行 onRestart 事件，之后会跳过 onCreate 事件，直接执行窗体的 onStart 事件。

3) void onResume()

onResume 事件在 onStart 事件之后执行。或者当前窗体被交换到后台后，在用户重新查看窗体时，窗体还没有被销毁，也没有执行过 onStop 事件(窗体还继续存在于 Task 中)，则会跳过窗体的 onCreate 和 onStart 事件，直接执行 onResume 事件。

4) void onPause()

onPause()用于暂停线程，例如有一个交通服务线程，我们不想在后台运行它，此时就可以使用 onPause()停止它。onPause 事件在窗体被交换到后台时执行。

5) void onStop()

onStop 事件在 onPause 事件之后执行。如果一段时间内用户还没有重新查看该窗体，则该窗体的 onStop 事件将会被执行；或者用户直接按了 Back 键，将该窗体从当前 Task 中移除，也会执行该窗体的 onStop 事件。

6) void onRestart()

onPause()事件在 onStop 事件执行后，如果窗体和其所在的进程没有被系统销毁，此时用户又重新查看该窗体，则会执行窗体的 onRestart 事件，onRestart 事件后会跳过窗体的 onCreate 事件直接执行 onStart 事件。

7) void onDestroy()

onDestroy 事件在 Activity 被销毁的时候执行。在窗体的 onStop 事件之后，如果没有再次查看该窗体，Activity 则会被销毁。

最后用一个实际的例子来说明 Activity 的各个生命周期。假设有一个程序由两个 Activity A 和 B 组成，A 是这个程序的启动界面。当用户启动程序时，Process 和默认的 Task 分别被创建，接着 A 被压入到当前的 Task 中，依次执行了 onCreate、onStart 和 onResume 事件，被呈现给了用户；此时用户选择 A 中的某个功能开启界面 B，界面 B 被压入当前 Task 遮盖住了 A，A 的

onPause 事件执行，B 的 onCreate、onStart 和 onResume 事件执行，并呈现界面 B 给用户；用户在界面 B 操作完成后，使用 Back 键回到界面 A，界面 B 不再可见，界面 B 的 onPause、onStop 和 onDestroy 事件执行，A 的 onResume 事件被执行，并呈现界面 A 给用户。此时突然来电，界面 A 的 onPause 事件被执行，电话接听界面被呈现给用户，用户接听完电话后，又按了 Home 键回到桌面，打开另一个程序“联系人”，添加了联系人信息又做了一些其他的操作，此时界面 A 不再可见，其 onStop 事件被执行，但并没有被销毁。此后用户重新从菜单中点击了我们的程序，由于 A 和其所在的进程和 Task 并没有被销毁，A 的 onRestart 事件和 onStart 事件被执行，接着 A 的 onResume 事件被执行，A 又被呈现给了用户。用户这次使用完后，按 Back 键返回到桌面，A 的 onPause、onStop 事件被执行，随后 A 的 onDestroy 事件被执行，由于当前 Task 中已经没有任何 Activity，A 所在的 Process 的重要程度被降到很低，很快 A 所在的 Process 被系统结束。

4.6　进程和线程的那些事儿

进程和线程是什么？就是你现在正在干什么！当打开电脑中的进程管理器后，会显示当前运行的所有程序。安卓中的进程和线程是怎样的呢？且听为师慢慢道来。当某个组件第一次运行的时候，Android 就启动了一个进程，默认的，所有的组件和程序运行在这个进程和线程中。当然，也可以安排组件在其他的进程或者线程中运行。

4.6.1　进程

组件运行的进程由 manifest file 控制。组件的节点(<activity>、<service>、<receiver>和<provider>)都包含一个 process 属性。这个属性可以设置组件运行的进程：可以配置组件在一个独立进程运行，或者多个组件在同一个进程运行。甚至可以多个程序在一个进程中运行——如果这些程序共享一个 User ID 并给定同样的权限。<application> 节点也包含 process 属性，用来设置程序中所有组件的默认进程。

所有的组件在此进程的主线程中实例化，系统对这些组件的调用从主线程中分离，并非每个对象都会从主线程中分离。一般来说，响应例如 View.onKeyDown()用户操作的方法和通知的方法也在主线程中运行。这就表示，组件被系统调用的时候不应该长时间运行或者阻塞操作(如网络操作或者计算大量数据)，因为这样会阻塞进程中的其他组件，可以把这类操作从主线程中分离。

当更加常用的进程无法获取足够的内存时，Android 可能会关闭不常用的进程，下次启动程序的时候会重新启动进程。

当决定哪个进程需要被关闭的时候，Android 会考虑哪个对用户更加有用，如 Android 会倾向于关闭一个长期不显示在界面的进程来支持一个经常显示在界面的进程。是否关闭一个进程决定于组件在进程中的状态。

4.6.2　线程

用户界面需要很快地对用户进行响应，因此某些费时的操作，如网络连接、下载或者非常占用服务器时间的操作应该放到其他线程。也就是说，即使为组件分配了不同的进程，有时候

也需要再分配线程。

线程是通过 Java 的标准对象 Thread 来创建的，Android 提供了很多方便的管理线程的方法，具体如下。

(1) Looper 在线程中运行的一个消息循环。

(2) Handler 传递一个消息。

(3) HandlerThread 创建一个带有消息循环的线程。

(4) Android 让一个应用程序在单独的线程中指导它创建自己的线程。

(5) 所有应用程序组件(Activity、Service、Broadcast Receiver)都在理想的主线程中实例化。

(6) 没有一个组件应该长时间执行或是阻塞操作(例如网络呼叫或是计算循环)，当被系统调用时，这将中断所有在该进程的其他组件。

(7) 可以创建一个新的线程来执行长期操作。

4.6.3 远程调用

Android 有一个远程调用(RPCs)的轻量级机制，通过这个机制，可以在本地调用实现应用功能的方法。在远程执行(在其他进程执行)，还可以返回一个值。要实现这个需求，必须分解方法调用，并且所有要传递的数据必须是操作系统可以访问的级别。从本地的进程和内存地址传送到远程的进程和内存地址并在远程处理和返回，返回值必须向相反的方向传递。Android 提供了做以上操作的代码，所以开发者可以专注于实现 RPC 的接口。

一个 RPC 接口只能包含方法，所有的方法都是同步执行的(直到远程方法返回，本地方法才结束阻塞)，没有返回值的时候也是如此。

简单说，此机制的描述如下：使用 IDL(Interface Definition Language)定义你想要实现的接口，aidl 工具可以生成用于 Java 的接口定义，本地和远程都要使用这个定义，它包含两个类，具体关系如图 4-5 所示。

其中，inner 类包含了所有的管理远程程序(符合 IDL 描述的接口)所需要的代码。所有的 inner 类实现了 IBinder 接口。其中一个在本地使用，可以不管它的代码；另外一个叫做 Stub 的类继承了 Binder 类。为了实现远程调用，类 Stub 包含 RPC 接口。开发者可以继承 Stub 类来实现需要的方法。

一般来说，远程进程会被一个 Service 管理(因为 Service 可以通知操作系统这个进程的信息并和其他进程通信)，它也会包含 aidl 工具产生的接口文件，Stub 类实现了我们应用程序中的方法。服务的客户端只需要 aidl 工具产生的接口文件。

如何连接服务和客户端调用的方法如下。

- 服务的客户端(本地)会实现 onServiceConnected()和 onServiceDisconnected()方法，这样，当客户端连接或者断开连接的时候可以获取到通知。通过 bindService()获取到服务的连接。
- 服务的 onBind()方法中可以接收或者拒绝连接，这取决于它收到的 intent (intent 通过 bindService()方法连接到服务)。如果服务接收了连接，会返回一个 Stub 类的实例。
- 如果服务接受了连接，Android 会调用客户端的 onServiceConnected()方法，并传递一个 Ibinder 对象(系统管理的 Stub 类的代理)，通过这个代理，客户端可以连接远程的服务。

以上的描述省略了很多 RPC 的机制。如果想进一步了解，请读者参见 Designing a Remote Interface Using AIDL 和 IBinder 类。

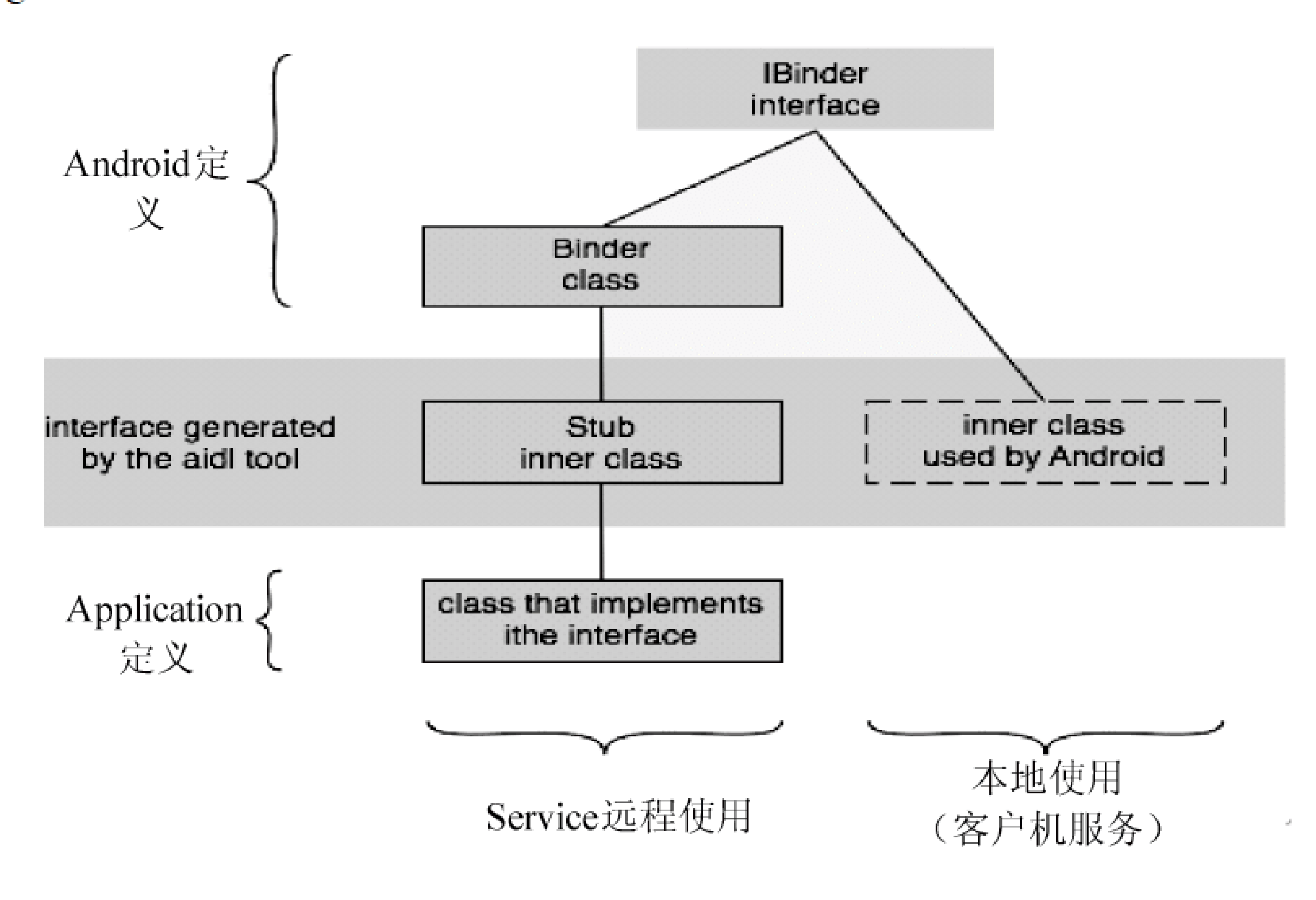


图 4-5　类关系

4.7　师傅的例子

虽然连续几天的面壁修炼学到了很多东西，但是 Activity、Intent、Service、BroadcastIntentReceiver 之间的关系太复杂，感觉自己并没有真正掌握，我决定向师傅请教。师傅为了更好的说明问题，通过一个例子进行了讲解。

隔壁二牛给他的孩子摆满月酒，二牛摆了十来桌，全村人都去了。但是你到哪桌呢？你可能会等他们坐的差不多了，然后看哪桌上你的知己多一点；也可能是提前跟你的那些知己商量好坐同一桌。安卓中的 Activity 就是一张饭桌，是二牛这十来桌中的一张桌子；Service 也很容易理解，桌子上的菜是怎么做的呢？我们不知道，只知道是厨房里的厨师做的，Service 做菜属于后台实现的，没有被摆到台面上；Intent and Intent Filters 相当于二牛家的管家或帮忙的，如果发现客人之中有实在不知坐在哪儿的，他会提议你坐在哪一桌，此时你还不能反驳，只能坐在那儿；BroadcastIntentReceiver 是一个广播，酒席开始之前，会让德高望重的村长讲几句话，他讲话的内容在座的人全都听见了；ContentProvider 犹如给二牛帮忙的人，现代社会称之为秘书，他将当天发生的点点滴滴都记录下来，例如王大婶家给了两只鸡，张大娘给了两斤鸡蛋……ContentProvider 就是一个记录本，保存了二牛当天满月酒的信息资料。

为师的这个比喻，虽然有点牵强，但是能说明它们之间的原理和运作流程了。由于本部分心法的重要性，因此建议你们继续练习。



第 5 章　一本秘籍闯天涯

组件是编程中的重要组成部分，一个项目通常由多个组件共同构成实现某项具体的功能。在 Android SDK 中，可以通过大量的组件来实现具体项目的需求。本章将详细介绍 Android 基本组件的知识，并通过具体实例的实现过程讲解各个组件的使用方法，为读者步入本书后面知识的学习打下坚实的基础。

5.1　下山的喜悦

今天和往常一样，迎着清晨的第一缕阳光踏上了习武场。虽然每天泼洒汗水，但是心头充满希望。刚刚吃完早饭，师傅将我叫到了藏经阁。师傅说江湖磨炼是检验修行水平的最佳途径，在下山之前师傅将本派秘籍《安卓百科全书》交给了我，说当遇到问题时可以从秘籍中找到解决问题的答案。

5.2　用 UI 配置行头

师傅建议我准备一身像样的行头，我来到离缥缈峰最近的一个镇上。选了一家最大的裁缝店，定做了一身侠客装，因为经费有限做的衣服总感觉低人一等，实在找不到更好的解决办法，无奈之下只好拿出秘籍《安卓百科全书》。翻阅之后顿时豁然开朗。行头装备不是一日置办成的，罗马城不是一日建成的，手机界面不是简单布置就能行的！看来手机界面的布局还颇有讲究，我决定好好研究一番。

5.2.1　“爷爷”级的 View 视图组件

类 View 是一个最基本的 UI 类，因为几乎所有的 UI 组件都是继承 View 来实现的，所以称之为“爷爷”级，其主要功能如下。

(1)　为指定的屏幕矩形区域存储布局和内容。

(2)　处理屏幕的尺寸和布局，绘制，焦点改变，翻屏，按键，手势。

(3) widget 基类。

类 View 的使用格式如下：

```
Android.view.View
```

Android 中常用的 View 类如表 5-1 所示。

表 5-1　View 类

文本(TextView)	输入框(EditText)
输入法(InputMethod)	活动方法(MovementMethod)
复选框(Checkbox)	滚动视图(ScrollView)
按钮(Button)	单选按钮(RadioButton)

5.2.2　Viewgroup 是一个大容器

Viewgroup 是一个大容器，能够对它里面的 View 进行布局处理。Viewgroup 的使用格式如下：

```
Android.view.Viewgroup
```

Viewgroup 用于包含并管理下级系列的 Views 和其他 Viewgroup，是一个布局的基类。Viewgroup 好像一个 View 容器，负责对添加进来的 View 进行布局处理。一个 Viewgroup 可以放到另一个 Viewgroup 中去，这是因为 Viewgroup 也是继承 GF View.Viewgroup 类，即是其他容器类的基类。Viewgroup 类之间的关系如图 5-1 所示。

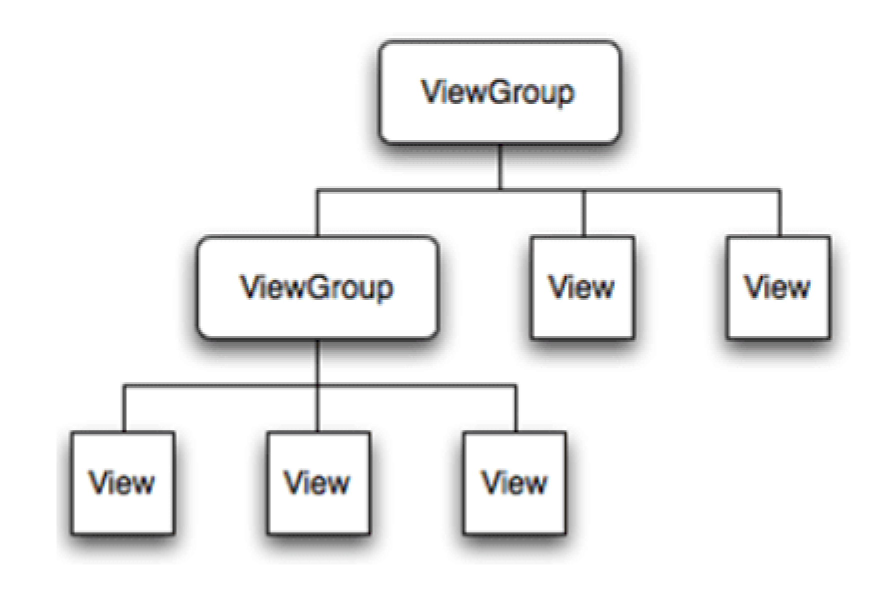


图 5-1　各类的继承关系

5.2.3　通过 Layout 来规划布局

布局就像容器，里面可以装下很多控件，而布局的作用就是对这些控件进行布局。在布局里面还可以套用其他的布局，这样可以实现界面的多样化和设计的灵活性。布局组件 Layout 的使用格式如下：

```
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
   Android:orientation="vertical"
   Android:layout_width="fill_parent"
   Android:layout_height="fill_parent"
>
```

一个布局容器里可以包括零个或多个布局容器，常用的 Layout 实现类有以下 5 个。

(1)　AbsoluteLayout：可以让子元素指定准确的 x、y 坐标值，并显示在屏幕上。(0, 0)为左上角，当向下或向右移动时，坐标值将变大。AbsoluteLayout 没有页边框，允许元素之间互相重叠(尽管不推荐)。我们通常不推荐使用 AbsoluteLayout，除非你有正当理由要使用它，因为它使界面代码太过刚性，以至于在不同的设备上可能不能很好地工作，效果如图 5-2 所示。

(2)　TableLayout：用于把子元素放入行与列中，不显示行、列或是单元格边界线，并且单元格不能横跨行，如 HTML 中一样，效果如图 5-3 所示。



图 5-2　AbsoluteLayout 效果

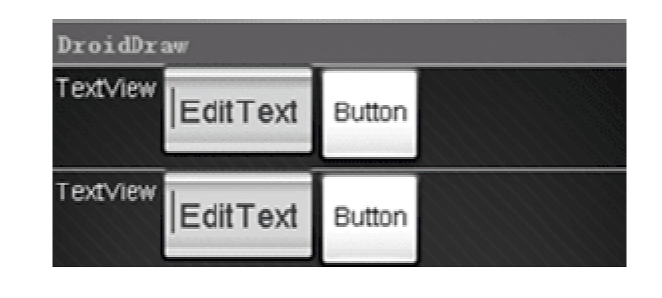


图 5-3　TableLayout 效果

(3)　FrameLayout：是最简单的一个布局对象。它被定制为屏幕上的一个空白备用区域，之后可以在其中填充一个单一对象，比如一张要发布的图片。所有的子元素将会固定在屏幕的左上角；你不能为 FrameLayout 中的一个子元素指定一个位置。后一个子元素将会直接在前一个子元素上进行覆盖填充，把它们部分或全部挡住(除非后一个子元素是透明的)。

(4)　RelativeLayout：允许子元素指定它们相对于其他元素或父元素的位置(通过 ID 指定)。因此，你可以以右对齐，或上、下对齐，或置于屏幕中央的形式来排列两个元素。元素按顺序排列，因此，如果第一个元素在屏幕的中央，那么相对于这个元素的其他元素将以屏幕中央的相对位置来排列。如果使用 XML 来指定这个 Layout，在定义它之前，被关联的元素必须定义，其效果结构如图 5-4 所示。

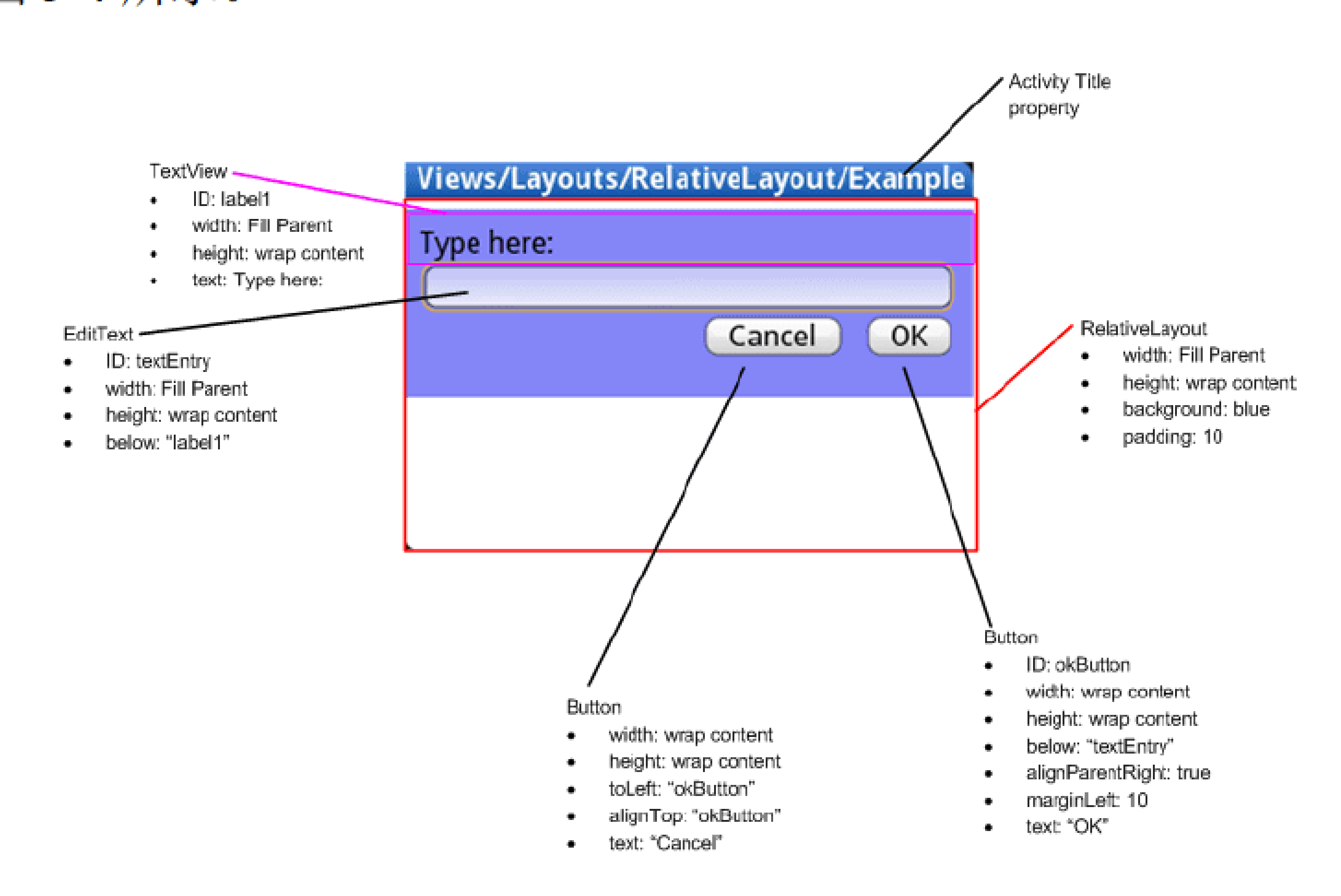


图 5-4　RelativeLayout 效果结构

(5)　LinearLayout：用于在一个方向上(垂直或水平)对齐所有子元素。所有子元素可以一个一个地堆放；也可以一个垂直列表每行只有一个子元素(无论它们有多宽)，如图 5-5 所示；也可以一个水平列表只是一列的高度(最高子元素的高度来填充)，如图 5-6 所示。

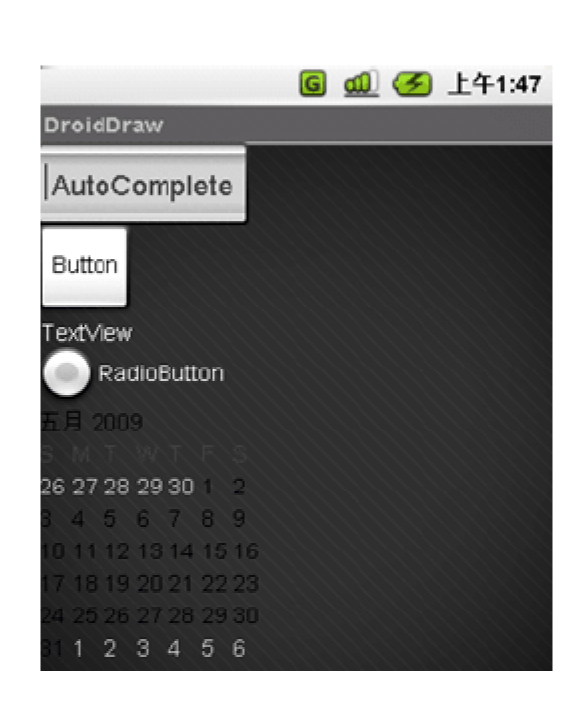


图 5-5　垂直布局

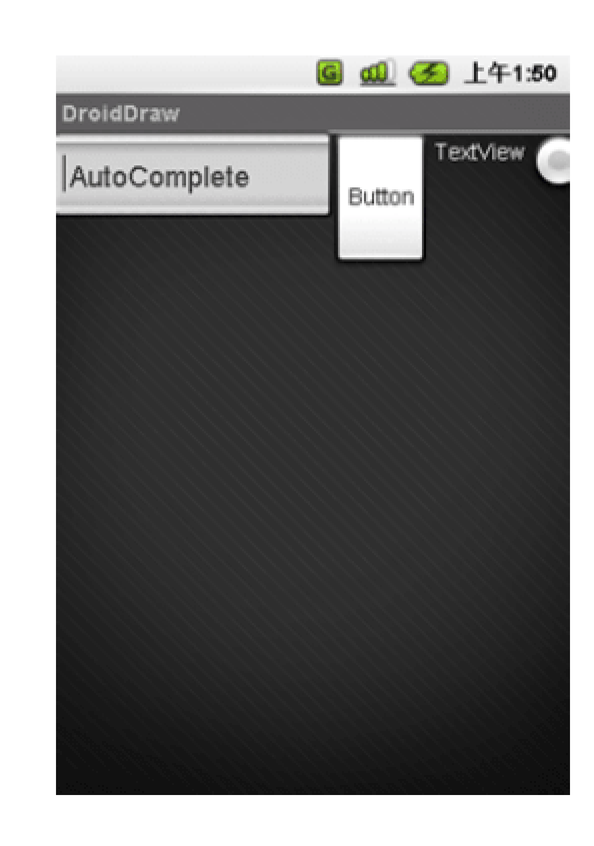


图 5-6　水平布局

5.2.4　LayoutParams 参数的意义

当一个 View 加入到一个 Viewgroup 后，假设加入到了 RelativeLayout 里面，你知道此时这个 View 在 RelativeLayout 里面是怎样显示的吗？究竟是显示在左边还是右边？上边还是下边？答案其实很简单：当向 Relative Layout 里面加入 View 时，我们传递一组值，并将这组值封装在 LayoutParams 类中。这样当再显示这个 View 时，其容器会根据封装在 LayoutParams 类的值来确认此 View 的显示大小和位置。由此可以看出，LayoutParams 的功能描述如下。

- 每一个 Viewgroup 类使用一个继承于 ViewGroup.LayoutParams 的嵌套类。
- 包含定义了子节点 View 尺寸和位置的属性类型。

LayoutParams 的具体结构图如图 5-7 所示。

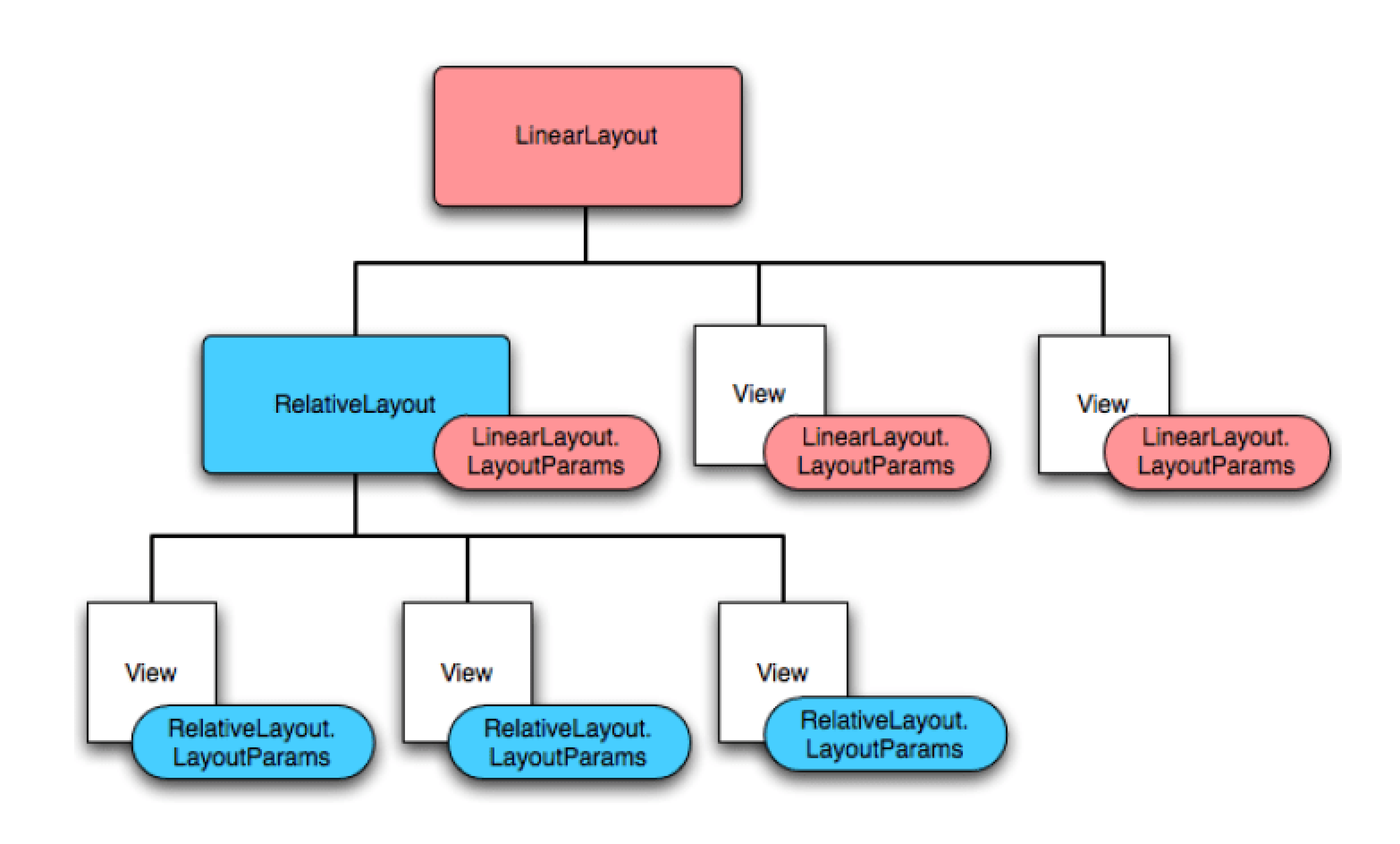


图 5-7　LayoutParams 结构图

5.2.5　小试牛刀

《安卓百科全书》对行头的置办和手机界面布局方法介绍得很详细，弄明白了 View、Viewgroup、Layout 和 LayoutParams 参数的基本关系后，我决定亲自试验一番。

题　目	目　的	源码路径
演练 1	实战演练基本布局控件的用法	“光盘\daima\6\widges”文件夹

为了保证此演练必胜，我在开始之前做了充分的准备，对整个过程和功能进行了详细的规划，具体功能和流程如图 5-8 所示。

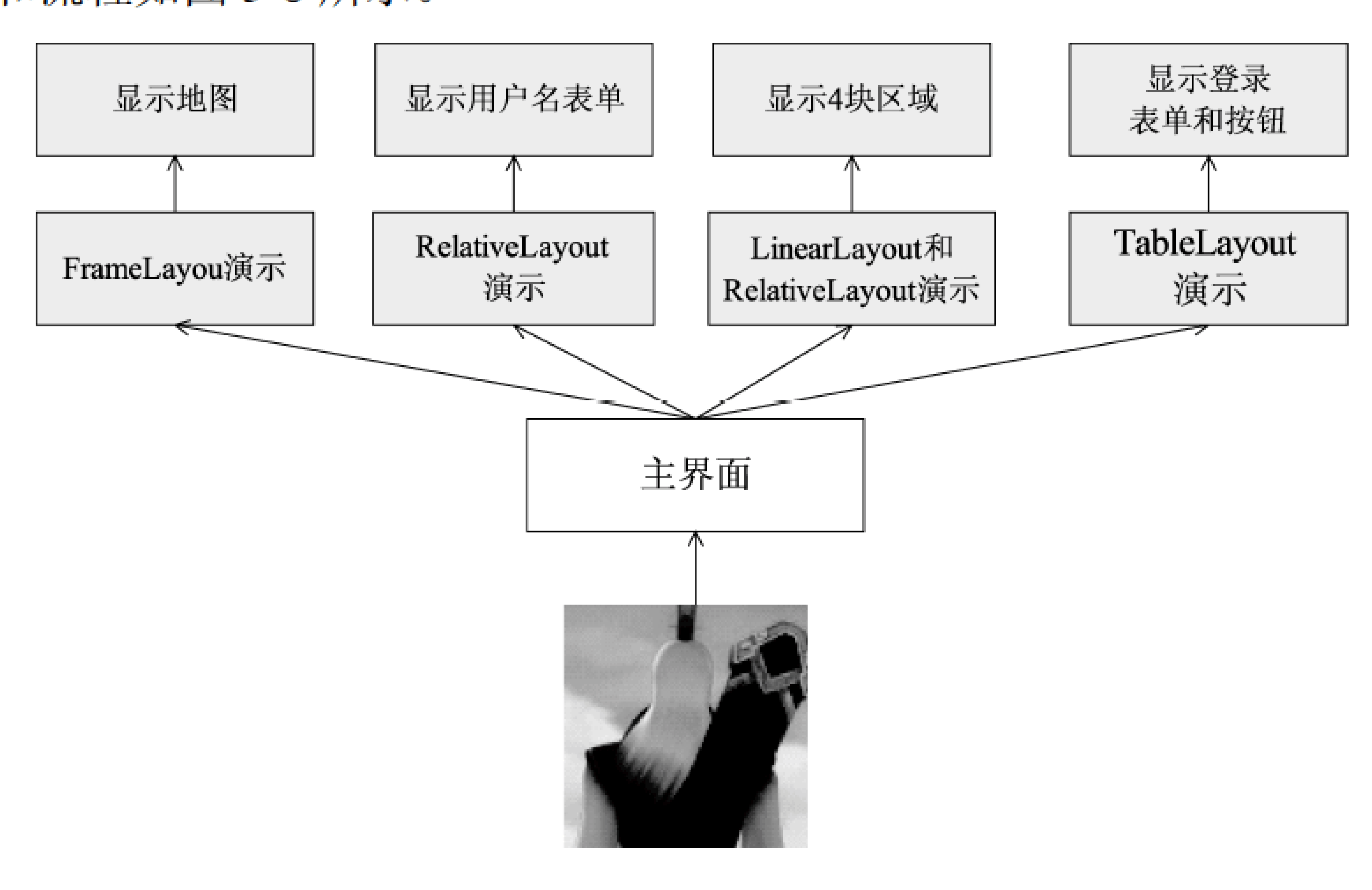


图 5-8　功能和流程

1. 新建工程

打开 Eclipse，依次选择 File | New | Android Project 菜单命令，新建一个名为“widges”的工程文件，如图 5-9 所示。

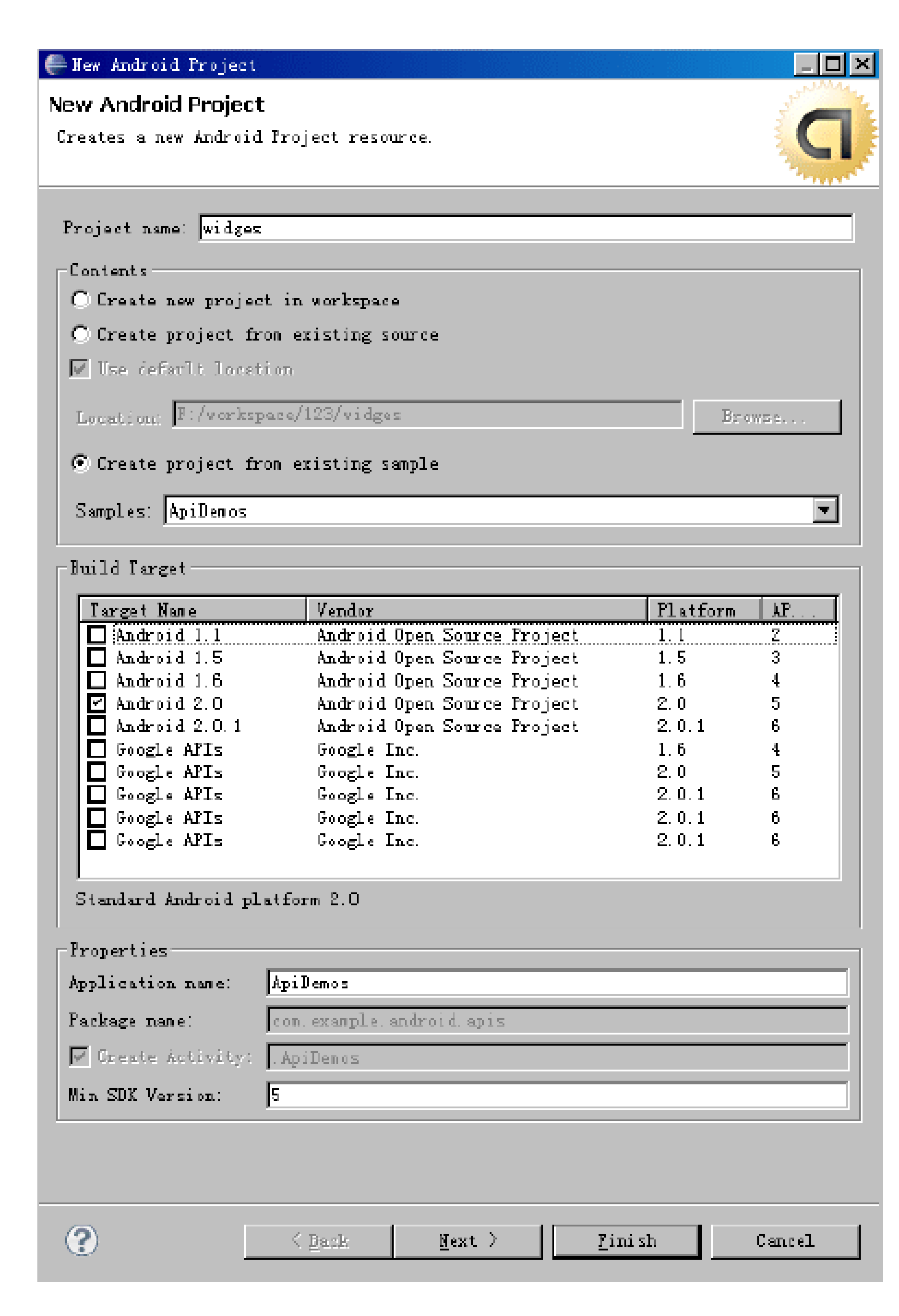


图 5-9　新建名为“widges”的工程文件

2. 编写代码

文件 Activity Main.java.java(路径为：src\com\eoeAndroid\layout\ActivityMain.java.java)是此项目的主要文件，用于调用各个公用文件来实现具体的功能，具体代码如下所示：

```
package com.eoeAndroid.layout;

import Android.app.Activity;
import Android.content.Intent;
import Android.os.Bundle;
import Android.view.View;
import Android.view.View.OnClickListener;
import Android.widget.Button;

public class ActivityMain extends Activity {
    OnClickListener listener0 = null;
    OnClickListener listener1 = null;
    OnClickListener listener2 = null;
    OnClickListener listener3 = null;
    Button button0;
    Button button1;
    Button button2;
    Button button3;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        listener0 = new OnClickListener() {
            public void onClick(View v) {
                Intent intent0 = new Intent(ActivityMain.this, ActivityFrameLayout.
                class);
                setTitle("FrameLayout");
                startActivity(intent0);
            }
        };
        listener1 = new OnClickListener() {
            public void onClick(View v) {
                Intent intent1 = new Intent(ActivityMain.this, ActivityRelativeLayout.
                class);
                startActivity(intent1);
            }
        };
        listener2 = new OnClickListener() {
            public void onClick(View v) {
                setTitle("在 ActivityLayout");
                Intent intent2 = new Intent(ActivityMain.this, ActivityLayout.
                class);
                startActivity(intent2);

            }
        };
```

```
        listener3 = new OnClickListener() {
            public void onClick(View v) {
                setTitle("TableLayout");
                Intent intent3 = new Intent(ActivityMain.this, ActivityTableLayout.
                class);
                startActivity(intent3);

            }
        };
        setContentView(R.layout.main);
        button0 = (Button) findViewById(R.id.button0);
        button0.setOnClickListener(listener0);
        button1 = (Button) findViewById(R.id.button1);
        button1.setOnClickListener(listener1);
        button2 = (Button) findViewById(R.id.button2);
        button2.setOnClickListener(listener2);
        button3 = (Button) findViewById(R.id.button3);
        button3.setOnClickListener(listener3);
    }
}
```

在上述代码中，函数 setContentView(R.layout.main)用于实现 Activity 和 main.xml 的关联；button0、button1、button2、button3 代表了 4 个 Button 按钮，在上述代码中对这 4 个按钮实现了引用，并给 Button 设置了单击监听器，每一个监听器都跳转到一个新的 Activity。

3. 布局界面

界面布局功能由文件 main.xml 实现，具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout width="fill parent"
    android:layout_height="fill_parent">
    <Button android:id="@+id/button0"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="FrameLayou 演示" />
    <Button android:id="@+id/button1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="RelativeLayout 演示" />
    <Button android:id="@+id/button2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="LinearLayout 和 RelativeLayout 演示" />
    <Button android:id="@+id/button3"
        android:layout width="fill parent"
        android:layout_height="wrap_content"
        android:text="TableLayout 演示" />
</LinearLayout>
```

上述代码是一个典型的 LinearLayout 布局样式。

4. 第一个按钮的处理动作

设计单击第一个按钮 button0 后的处理动作。在本实例中，单击第一个按钮 button0 后会显示一个地图，此界面是一个 FrameLayout 布局。在文件 activity_frame_layout.xml 中定义了这幅地图的显示样式，即在 FrameLayout 布局中添加了一个图片显示组件 ImageView 元素，具体实现代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout Android:id="@+id/left"
    xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:layout_width="fill_parent"                  /**x 轴方向填充空间*/
    Android:layout_height="fill_parent"                 /**y 轴方向填充空间*/
>
    <ImageView Android:id="@+id/photo"                  /**定义组件的 id*/
Android:src="@drawable/bg"
        Android:layout_width="wrap_content"             /**宽度能包容图片*/
        Android:layout_height="wrap_content"            /**高度能包容图片*/
  />
</FrameLayout>
```

在上述代码中，可以通过“Android:id”来访问定义的这个元素；“Android:layout_width="fill_parent"”表示 FrameLayout 布局可以在 x 轴方向填充的空间，“Android:layout_height="fill_parent"”表示 FrameLayout 布局可以在 y 轴方向填充的空间；“Android:layout_width="wrap_content"”和“Android:layout_height="wrap_content"”表示 ImageView 只需将图片完全包含即可。

5. 第二个按钮的处理动作

设计单击第二个按钮 button1 后的处理动作。在本实例中，单击第二个按钮 button1 后会显示要求输入用户名的表单，此功能是通过 RelativeLayout 实现的，对应文件 relative_layout.xml 的具体实现代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Demonstrates using a relative layout to create a form -->

<RelativeLayout
    xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:layout_width="fill_parent" Android:layout_height="wrap_content"
    Android:background="@drawable/blue" Android:padding="10dip">

    <TextView Android:id="@+id/label" Android:layout_width="fill_parent"
        Android:layout_height="wrap_content" Android:text="请输入用户名：" />
    <!--
        这个 EditText 放置在上边 id 为 label 的 TextView 的下边
    -->
    <EditText Android:id="@+id/entry" Android:layout_width="fill_parent"
        Android:layout_height="wrap_content"
        Android:background="@Android:drawable/editbox_background"
        Android:layout_below="@id/label" />

    <!--
```

```
        取消按钮和容器的右边齐平，并且设置左边的边距为 10dip
    -->
    <Button Android:id="@+id/cancel" Android:layout_width="wrap_content"
        Android:layout_height="wrap_content" Android:layout_below="@id/entry"
        Android:layout_alignParentRight="true"
        Android:layout_marginLeft="10dip" Android:text="取消" />

    <!--
        确定按钮在取消按钮的左侧，并且和取消按钮的高度齐平
    -->
    <Button Android:id="@+id/ok" Android:layout_width="wrap_content"
        Android:layout height="wrap content"
        Android:layout toLeftOf="@id/cancel"
        Android:layout_alignTop="@id/cancel" Android:text="确定" />
</RelativeLayout>
```

在上述代码中，主要参数的具体说明如下。

(1) Android:id：定义组件的 ID。

(2) Android:layout_width：设置组件的宽度，主要有如下两种方式。

- fill_parent：填充父容器。
- wrap_content：用于包含控件中的内容即可。

(3) Android:layout_height：定义组件的高度。

(4) Android:background="@drawable/blue"：定义组件的背景，在此设置了背景颜色。

(5) Android:padding="10dip"：“dip”表示依赖于设备的像素，有以下两种表现方式。

- padding：填充。
- margin：边框。

(6) Android:layout_below="@id/label"：将此组件放置于 ID 为 label 的组件的下方。此种方式是经典的布局方式，这种方式的好处是不用关心具体的细节，并且适配性很强，在不同屏幕、不同手机设备上都是通用的。

(7) Android:layout_alignParentRight="true"：也是相对布局，表示和父容器的右边对齐。

(8) Android:layout_marginLeft="10dip"：设置 ID 为 cancel 的 Button 的左边距为 10dips。

(9) Android:layout_toLeftOf="@id/cancel"：设置此组件在 ID 为 cancel 的组件的左边。

(10) Android:layout_alignTop="@id/cancel"：设置此组件同 ID 为 cancel 的组件的高度对齐。

6. 第三个按钮的处理动作

设计单击第三个按钮 button2 后的处理动作。在本实例中，单击第三个按钮 button2 后会显示一系列的文本，此功能是通过 LinearLayout 和 RelativeLayout 联合实现的，具体实现流程如下。

(1) 第 a 组第 a 项和第 a 组第 b 项：通过 RelativeLayout 实现的，此布局功能是通过文件 left.xml 定义的，具体实现代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
Android:id="@+id/left"
    xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:layout_width="fill_parent"
```

```
    Android:layout height="fill parent">
    <TextView Android:id="@+id/view1" Android:background="@drawable/blue"
        Android:layout_width="fill_parent"
        Android:layout_height="50px" Android:text="第 a 组第 a 项" />
    <TextView Android:id="@+id/view2"
        Android:background="@drawable/yellow"
        Android:layout_width="fill_parent"
        Android:layout_height="50px" Android:layout_below="@id/view1"
        Android:text="第 a 组第 b 项" />
</RelativeLayout>
```

在上述代码中，使用了两个 TextView，高度都是 50 像素。此处 TextView 的具体说明如下。

第一个 TextView：通过“@drawable/blue”设置其背景颜色是“blue”。

第二个 TextView：通过“Android:layout_below="@id/view1"”设置其位置位于第一个 TextView 的下方。

(2) 第 b 组第 a 项和第 b 组第 b 项：通过另外一个 RelativeLayout 实现的，此布局功能是通过文件 right.xml 定义的，具体实现代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout Android:id="@+id/right"
    xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:layout_width="fill_parent"
    Android:layout_height="fill_parent">
    <TextView Android:id="@+id/right_view1"
        Android:background="@drawable/yellow"
Android:layout_width="fill_parent"
        Android:layout_height="wrap_content" Android:text="第 b 组第 a 项" />
    <TextView Android:id="@+id/right_view2"
        Android:background="@drawable/blue"
        Android:layout_width="fill_parent"
        Android:layout_height="wrap_content"
        Android:layout_below="@id/right_view1" Android:text="第 b 组第 b 项" />
</RelativeLayout>
```

上述代码和文件 left.xml 类似，在此将不再进行详细介绍。

(3) 实现 Layout 和 Activity 的关联：即实现一个 Layout 和一个 Activity 的关联，而此 Layout 是在 XML 文件中被定义的。在 Activity 中，为使用方便可以自行构建一个 Layout。根据上述描述编写文件 ActivityLayout.java，具体实现代码如下所示：

```
public class ActivityLayout extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
       /**创建一个 Layout **/
        LinearLayout layoutMain = new LinearLayout(this);
        layoutMain.setOrientation(LinearLayout.HORIZONTAL);
        /**实现 Layout 和 Activity 的关联**/
        setContentView(layoutMain);
        /**得到一个 LayoutInflater 对象，此对象可以对 XML 布局文件进行解析，并生成一个
        view**/
```

```
        LayoutInflater inflate = (LayoutInflater) getSystemService(Context.
LAYOUT_INFLATER_SERVICE);
        RelativeLayout layoutLeft = (RelativeLayout) inflate.inflate(
                R.layout.left, null);
        RelativeLayout layoutRight = (RelativeLayout) inflate.inflate(
                R.layout.right, null);
      /**生成一个可以供 Layout 使用的 LayoutParams**/
        RelativeLayout.LayoutParams relParam = new RelativeLayout.LayoutParams(
                RelativeLayout.LayoutParams.WRAP_CONTENT,
                RelativeLayout.LayoutParams.WRAP_CONTENT);
/**将 layoutLeft 添加到 layoutMain，第一个参数是添加进去的 view，第二、三个分别是 view 的
高度和宽度**/
        layoutMain.addView(layoutLeft, 100, 100);
/**将 layoutRight 添加到 layoutMain，第二个参数是一个 RelativeLayout.LayoutParams**/
        layoutMain.addView(layoutRight, relParam);
    }
```

7. 第 7 个按钮的处理动作

设计单击第四个按钮 button3 后的处理动作。在本实例中，单击第四个按钮 button3 后会显示一个整齐排列的表单，此功能是通过 TableLayout 实现的，对应文件 activity_table_layout.xml 的具体实现代码如下：

```
<TableLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:layout_width="fill_parent" Android:layout_height="fill_parent"
    Android:stretchColumns="1">
    <TableRow>
        <TextView Android:text="用户名:" Android:textStyle="bold"
            Android:gravity="right" Android:padding="3dip" />
        <EditText Android:id="@+id/username" Android:padding="3dip"
            Android:scrollHorizontally="true" />
    </TableRow>
    <TableRow>
        <TextView Android:text="密码:" Android:textStyle="bold"
            Android:gravity="right" Android:padding="3dip" />
        <EditText Android:id="@+id/password" Android:password="true"
            Android:padding="3dip" Android:scrollHorizontally="true" />
    </TableRow>
    <TableRow Android:gravity="right">
        <Button Android:id="@+id/cancel"
            Android:text="取消" />
        <Button Android:id="@+id/login"
            Android:text="登录" />
    </TableRow>
</TableLayout>
```

在上述代码中，首先通过标签“TableLayout”定义了一个表格布局，然后通过“TableRow”标签定义了表格布局里的一行，用户可以根据需要在每一行中加入自己需要的一些组件。

8. 测试

经过一上午的努力，整个演练进行得十分顺利，下面是我的具体测试流程。

第 1 步：在 Eclipse 中打开刚编写的项目文件，右键单击项目名“widges”，在弹出的菜单

中选择 Run As→Android Application 命令后开始编译运行当前项目，如图 5-10 所示。

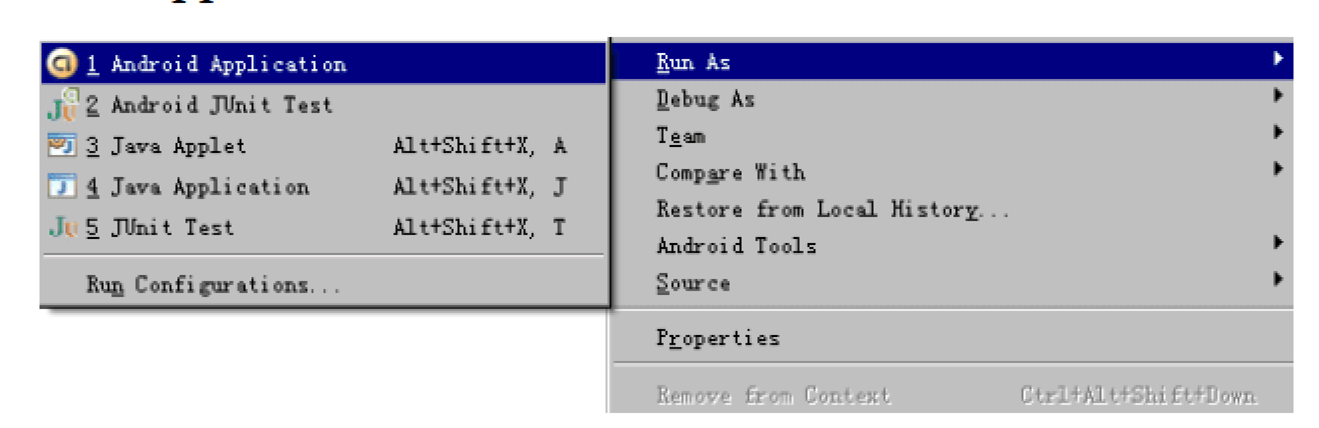


图 5-10　选择命令

第 2 步：运行后的初始效果如图 5-11 所示。

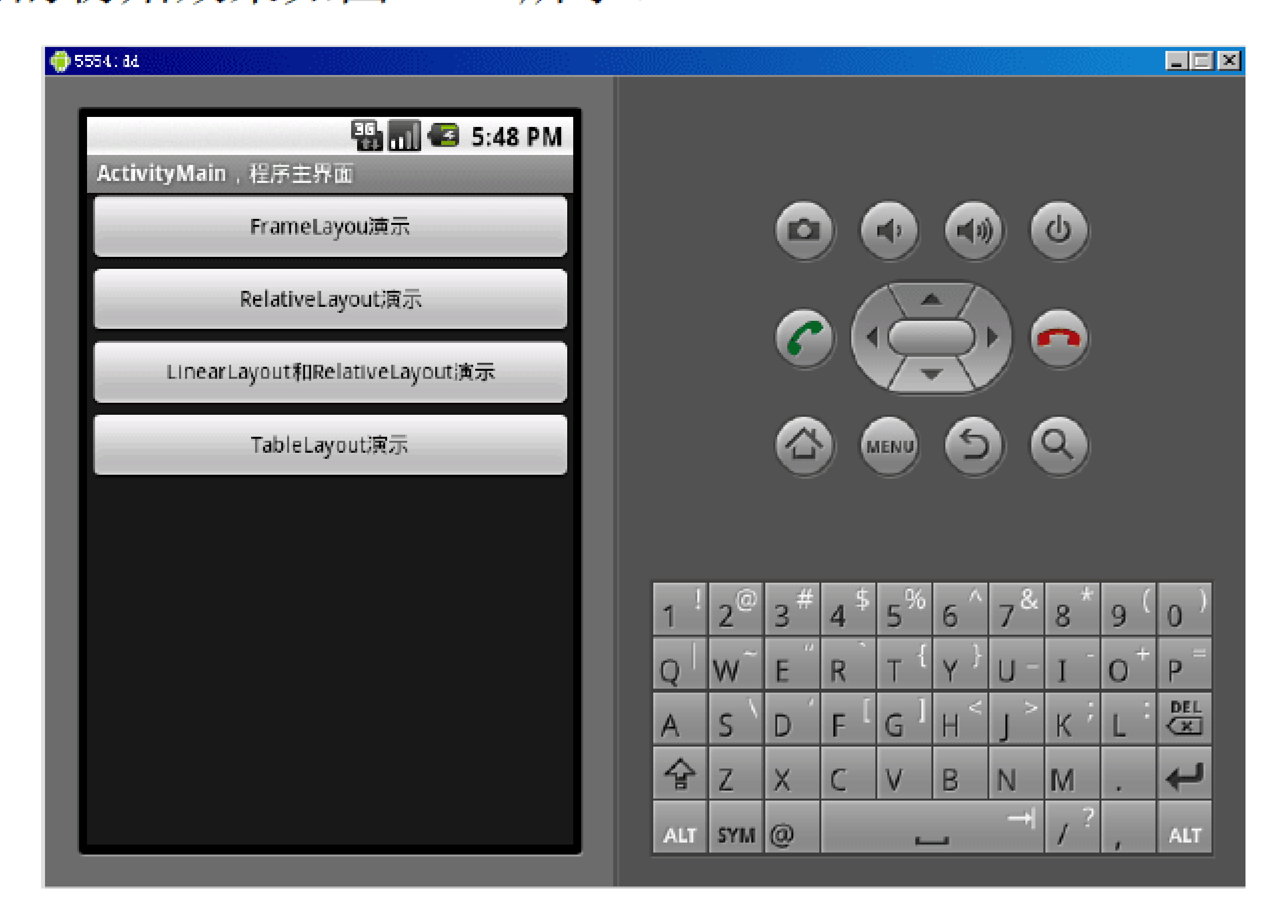


图 5-11　初始效果

第 3 步：单击“FrameLayout 演示”按钮后会显示指定的图片，效果如图 5-12 所示。

第 4 步：单击“RelativeLayout 演示”按钮后会显示输入用户名，效果如图 5-13 所示。

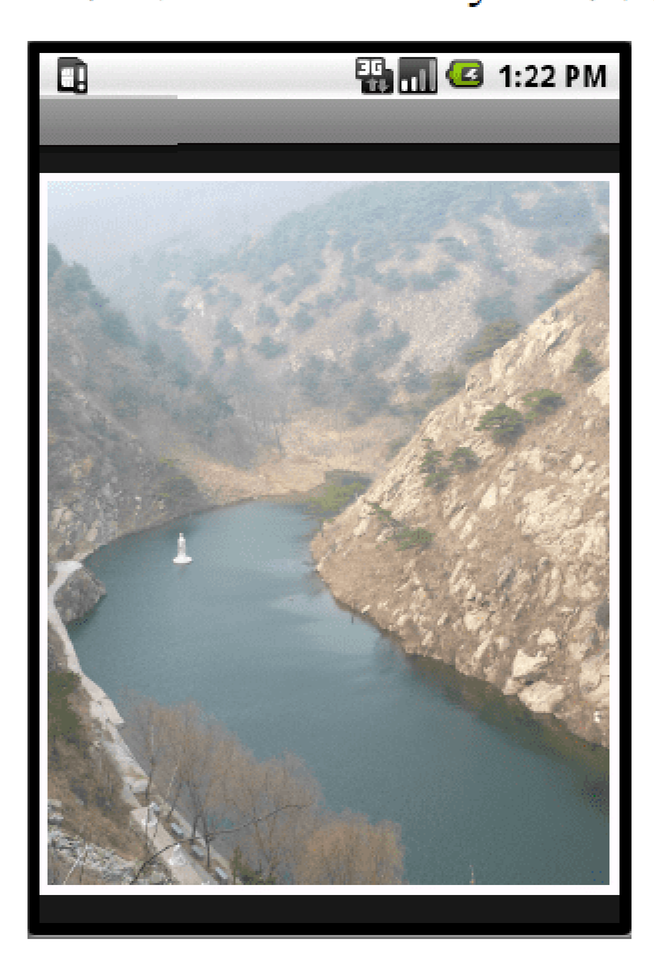


图 5-12　显示图片



图 5-13　显示输入用户名

第 5 步：单击“LinearLayout 和 RelativeLayout 演示”按钮后会显示 4 块不同样式的区域块，效果如图 5-14 所示。

第 6 步：单击“TableLayout 演示”按钮后会显示用户登录表单，效果如图 5-15 所示。

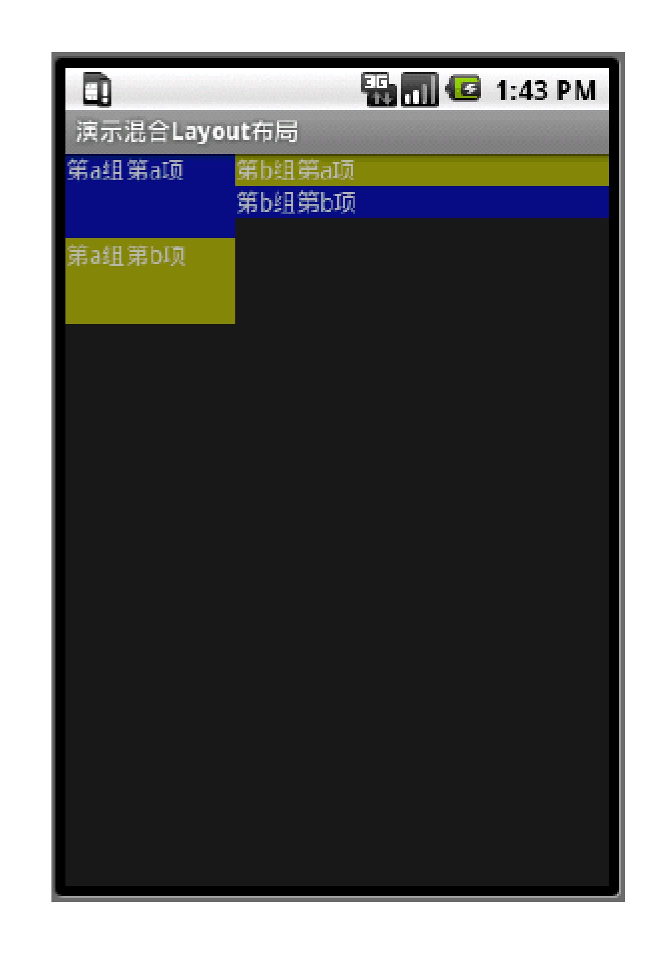


图 5-14　4 块不同样式的区域块



图 5-15　用户登录表单

5.3　布局我的侠客路

“少年听雨歌楼上，红烛昏罗帐；壮年听雨客舟中，江阔云低，断燕叫西风。”在酒馆中听着江湖曲，心中万千感慨。远离师傅和师兄的呵护，我独自一人仗剑走天涯。为了早日圆我的侠客梦，我决定好好布局一番。回到客栈房中，拿出《安卓百科全书》，只见上面写道：安卓中有五种界面布局对象，分别是 FrameLayout(框架布局)、LinearLayout(线性布局)、AbsoluteLayout(绝对布局)、RelativeLayout(相对布局)和 TableLayout(表格布局)。

5.3.1　纵览五大布局对象

在安卓中有五大布局对象，具体如图 5-16 所示。

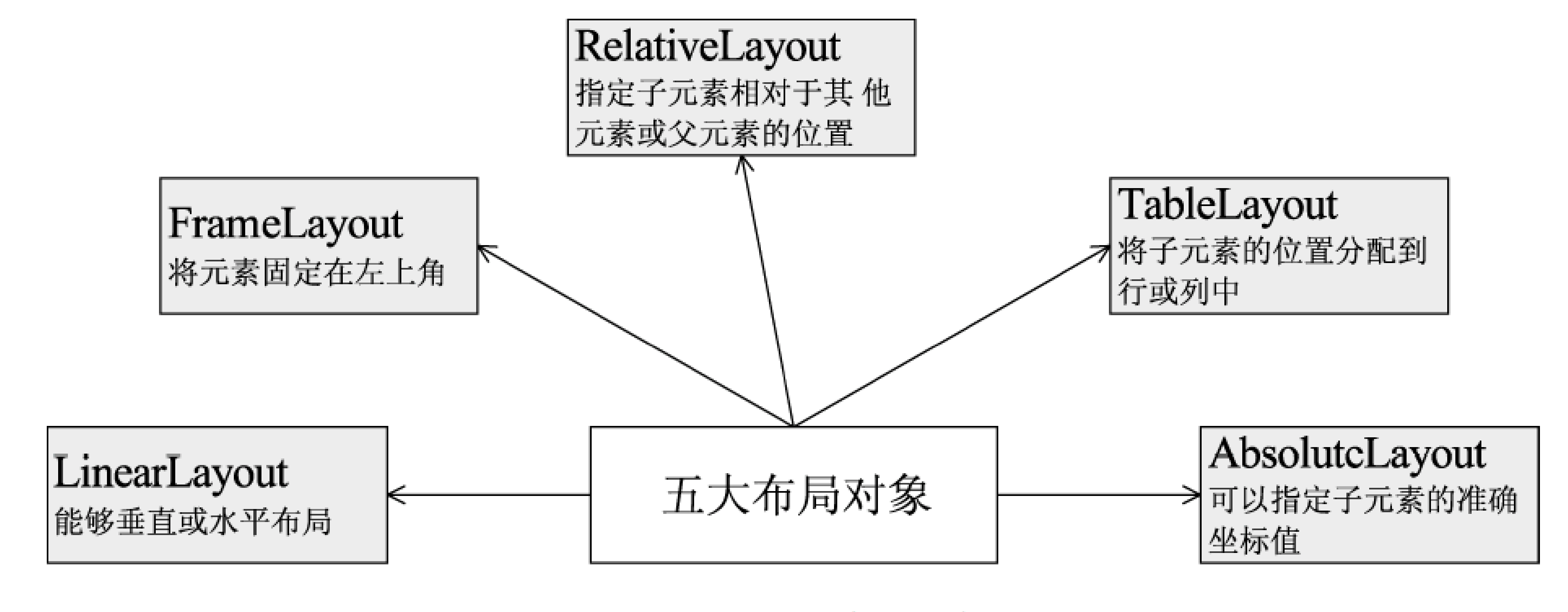


图 5-16　五大布局对象

1. LinearLayout

LinearLayout(线性布局)能够根据为它设置的垂直或水平属性值来排列所有的子元素。所有的子元素都被堆放在其他元素之后，因此，一个垂直列表的每一行只会有一个元素，不管它们有多宽，而一个水平列表将会只有一个行高(高度为最高子元素的高度加上边框高度)。

LinearLayout 保持子元素之间的间隔以及互相对齐(相对一个元素的右对齐、中间对齐或者左对齐)。

LinearLayout 还支持为单独的子元素指定 weight，其好处是允许子元素可以填充屏幕上的剩余空间。同时也避免了在一个大屏幕中一串小对象挤成一堆的情况，可以允许它们放大填充空白。子元素指定一个 weight 值，剩余的空间就会按照这些子元素指定的 weight 比例分配给这些子元素，默认的 weight 值为 0。假设有三个文本框，其中两个指定了 weight 值为 1，那么，这两个文本框将等比例地放大，并填满剩余空间，而第三个文本框不会放大。下面我们用事实来说话，具体的代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
   <LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="2">
    <TextView
        android:text="Welcome to Mr Wei's blog"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
     />
    </LinearLayout>
    <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout weight="1">

    <TextView
        android:text="red"
        android:gravity="center_horizontal" //这里字水平居中
        android:background="#aa0000"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
    <TextView
        android:text="green"
        android:gravity="center_horizontal "
        android:background="#00aa00"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
    </LinearLayout>
</LinearLayout>
```

上述代码使用了 LinearLayout 布局，执行后的效果如图 5-17 所示。

2. FrameLayout

FrameLayout(框架布局)是最简单的一个布局对象，它被定制为屏幕上的一个空白备用区域，之后可以在里面填充一个单一对象，例如一张图片。FrameLayout 会将所有的子元素固定在屏幕的左上角，我们不能设置 FrameLayout 中一个子元素的位置。后一个子元素将会直接在前一个子元素之上进行覆盖填充，把它们部分或全部挡住(除非后一个子元素是透明的)。具体的代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
 <!-- 我们在这里加了一个 Button 按钮 -->
<Button
    android:text="button"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
/>
<TextView
    android:text="textview"
    android:textColor="#0000ff"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>
</FrameLayout>
```

上述代码使用了 FrameLayout 布局，执行后的效果如图 5-18 所示。



图 5-17　LinearLayout 布局



图 5-18　FrameLayout 布局

3. AbsoluteLayout

AbsoluteLayout(绝对布局)可以指定其子元素的准确 x、y 坐标值，并显示在屏幕上。(0，0)为左上角，当向下或向右移动时，坐标值将随之变大。AbsoluteLayout 没有页边框，可以允许元素之间互相重叠。具体的代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```
    android:layout width="fill parent"
    android:layout_height="fill_parent"
    >
<EditText
    android:text="Welcome to Mr Wei's blog"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
/>
<Button
    android:layout_x="250px"                //设置按钮的 X 坐标
    android:layout_y="40px"                 //设置按钮的 Y 坐标
    android:layout_width="70px"             //设置按钮的宽度
    android:layout height="wrap content"
    android:text="Button"
/>
</AbsoluteLayout>
```

上述代码使用了 AbsoluteLayout 布局，执行后的效果如图 5-19 所示。



图 5-19　AbsoluteLayout 布局

注意： 在日常项目应用中不推荐使用 AbsoluteLayout，因为它会使界面代码太过刚性，以致于在不同的设备上可能不会很好地工作。

4. RelativeLayout

RelativeLayout(相对布局)，指定子元素相对于其他元素或父元素的位置(通过 ID 指定)。我们可以采用右对齐，或上、下对齐，或者置于屏幕中央的形式来排列两个元素。因为元素按顺序排列，因此如果第一个元素在屏幕的中央，那么相对于这个元素的其他元素将以屏幕中央的相对位置来排列。如果使用 XML 来指定这个 Layout，在定义它之前必须定义被关联的元素。具体的代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Welcome to Mr Wei's blog:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
```

```
        android:layout width="wrap content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```

上述代码使用了 RelativeLayout 布局，执行后的效果如图 5-20 所示。



图 5-20　RelativeLayout 布局

5. TableLayout

TableLayout(表格布局)，将子元素的位置分配到行或列中。一个 TableLayout 布局由许多的 TableRow 组成，每个 TableRow 都会定义一个 row。TableLayout 容器不会显示 row、cloumns 或单元格的边框线。每个 row 拥有零个或多个单元格；每个单元格拥有一个 View 对象。表格由列和行组成许多的单元格。表格允许单元格为空，单元格不能跨列。具体的代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView android:layout_column="1" android:text="Open..." />
        <TextView android:text="Ctrl-O" android:gravity="right" />
    </TableRow>
    <TableRow>
        <TextView android:layout_column="1" android:text="Save..." />
        <TextView android:text="Ctrl-S" android:gravity="right" />
    </TableRow>
    //这里是上图中的分隔线
    <View android:layout height="2dip" android:background="#FF909090" />
    <TableRow>
        <TextView android:text="X" />
        <TextView android:text="Export..." />
        <TextView android:text="Ctrl-E" android:gravity="right " />
    </TableRow>
    <View android:layout_height="2dip" android:background="#FF909090" />
    <TableRow>
        <TextView android:layout_column="1" android:text="Quit"
```

```
            android:padding="3dip" />
    </TableRow>
</TableLayout>
```

上述代码使用了 TableLayout 布局，执行后的效果如图 5-21 所示。



图 5-21　TableLayout 布局

5.3.2　演练垂直线性布局

题　目	目　的	源码路径
演练 2	实战演练垂直线性布局 Vertical 的用法	“光盘\daima\5\vertical”文件夹

第 1 步：打开 Eclipse，依次选择 File | New | Android Project 菜单命令，新建一个名为“vertical”的工程文件。

第 2 步：编写布局文件 main.xml，插入 4 个 TextView，分别用于显示“第 1 行”、“第 2 行”、“第 3 行”、“第 4 行”。具体的代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
     <TextView
        android:text="第 1 行"
        android:gravity="center_vertical"
        android:textSize="15pt"
        android:background="#aa0000"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>

     <TextView
        android:text="第 2 行"
        android:textSize="15pt"
        android:gravity="center vertical"
        android:background="#00aa00"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
     <TextView
        android:text="第 3 行"
        android:textSize="15pt"
        android:gravity="center_vertical"
        android:background="#0000aa"
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
    <TextView
        android:text="第 4 行"
        android:textSize="15pt"
        android:gravity="center_vertical"
        android:background="#aaaa00"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>

</LinearLayout>
```

第 3 步：编写文件 strings.xml，用于设置标题文本。其主要代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">一萧一剑</string>
    <string name="app_name">独饮江边</string>
</resources>
```

第 4 步：主文件 Activity01.java 是自动生成的，能够调用界面布局文件的样式在手机屏幕中显示。其主要代码如下所示：

```
package com.yarin.android.chuizhixian;

import android.app.Activity;
import android.os.Bundle;

public class Activity01 extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

这样整个演练就结束了，执行后的效果如图 5-22 所示。



图 5-22　执行效果

5.3.3 演练水平线性布局

题 目	目 的	源码路径
演练 3	实战演练水平线性布局 horizontal 的用法	“光盘\daima\5\horizontal”文件夹

第 1 步：打开 Eclipse，新建一个名为“horizontal”的工程文件。

第 2 步：编写布局文件 main.xml，插入 4 个 TextView，分别显示“1 列”、“2 列”、“3 列”、“4 列”。具体的代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:text="1 列"
        android:gravity="center_horizontal"
        android:background="#aa0000"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
    <TextView
        android:text="2 列"
        android:gravity="center horizontal"
        android:background="#00aa00"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
    <TextView
        android:text="3 列"
        android:gravity="center_horizontal"
        android:background="#0000aa"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
    <TextView
        android:text="4 列"
        android:gravity="center_horizontal"
        android:background="#aaaa00"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
</LinearLayout>
```

第 3 步：主文件 Activity01.java 是自动生成的，能够调用界面布局文件的样式在手机屏幕中显示。其主要实现代码如下所示：

```
public class Activity01 extends Activity
{
    /** Called when the activity is first created. */
    @Override
```

```
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

至此，整个演练就完美结束了，执行后的效果如图 5-23 所示。

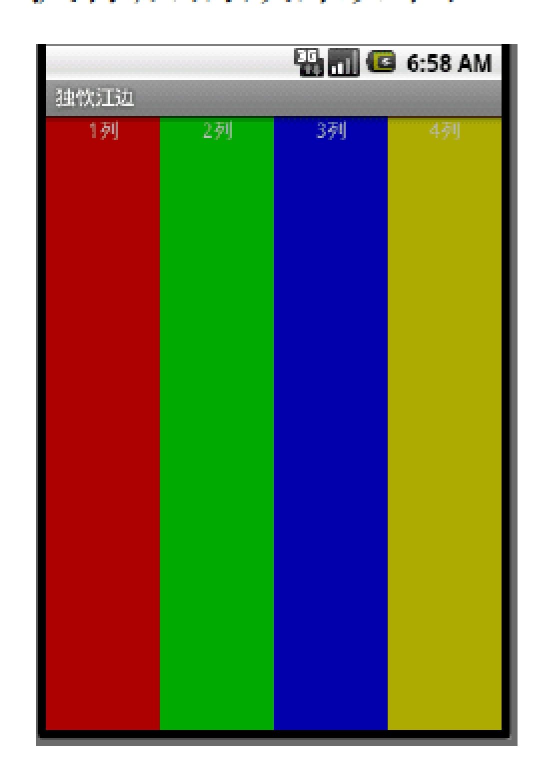


图 5-23　执行效果

5.3.4　演练相对布局

题　目	目　的	源码路径
演练 4	实战演练相对布局 RelativeLayout 的用法	“光盘\daima\5\RelativeLayout”文件夹

第 1 步：打开 Eclipse，新建一个名为“RelativeLayout”的工程文件。

第 2 步：编写布局文件 main.xml，分别插入一个 TextView、一个 EditText 和两个 Button。具体的代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <TextView
      android:id="@+id/label"
      android:layout width="fill parent"
      android:layout_height="wrap_content"
      android:text="写上您的祝福:"/>
   <EditText
      android:id="@+id/entry"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:background="@android:drawable/editbox_background"
      android:layout_below="@id/label"/>
   <Button
      android:id="@+id/ok"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="确定" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="取消" />
</RelativeLayout>
```

至此，整个演练就完美结束了，执行后的效果如图 5-24 所示。



图 5-24　执行效果

5.3.5　演练表单布局

题　目	目　的	源码路径
演练 5	实战演练表单布局 TableLayout 的用法	“光盘\daima\5\TableLayout”文件夹

第 1 步：打开 Eclipse，新建一个名为“TableLayout”的工程文件。

第 2 步：编写布局文件 main.xml，分别插入 7 个 TextView、6 个 TableLayout。具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="打开..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-O"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="保存..."
```

```
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-S"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="保存为..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-Shift-S"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
    <View
        android:layout_height="2dip"
        android:background="#FF909090" />
    <TableRow>
        <TextView
            android:text="*"
            android:padding="3dip" />
        <TextView
            android:text="导入..."
            android:padding="3dip" />
    </TableRow>
    <TableRow>
        <TextView
            android:text="*"
            android:padding="3dip" />
        <TextView
            android:text="导出..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-E"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
    <View
        android:layout_height="2dip"
        android:background="#FF909090" />
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="离开"
            android:padding="3dip" />
    </TableRow>
</TableLayout>
```

至此，整个演练就圆满结束了，执行后的效果如图 5-25 所示。

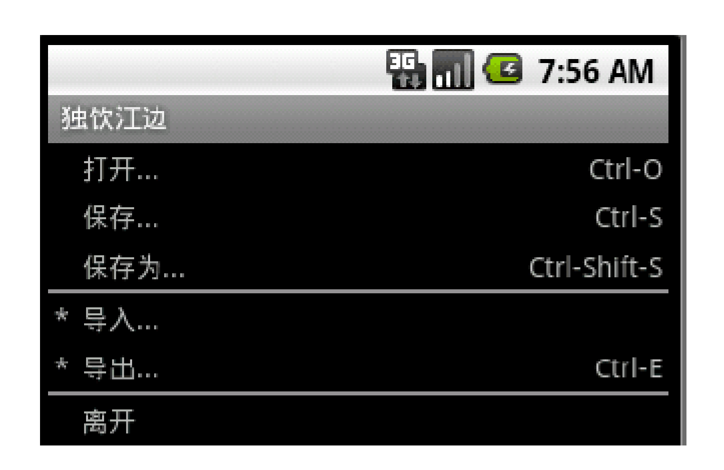


图 5-25　执行效果

5.3.6　演练切换卡

题　目	目　的	源码路径
演练 6	实战演练切换卡 TabWidget 的用法	“光盘\daima\5\TabWidget”文件夹

本节将通过一个具体实例的实现过程，讲解切换卡 TabWidget 的基本使用方法。本实例保存在“光盘\daima\5\TabWidget”文件夹中，本实例的具体实现流程如下。

第 1 步：打开 Eclipse，新建一个名为“TabWidget”的工程文件。

第 2 步：编写布局文件 main.xml，分别插入 3 个 TextView 和一个 TabWidget。具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent">
            <TextView
                android:id="@+id/textview1"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="这是一个 tab" />
            <TextView
                android:id="@+id/textview2"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="这是另一个 tab" />
```

```
          <TextView
              android:id="@+id/textview3"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:text="这是第三个 tab" />
    </FrameLayout>
   </LinearLayout>
</TabHost>
```

第 3 步：编写文件 Activity01.java，具体实现过程如下。

(1) 声明 TabHost 对象，并获取 TabHost 对象。

(2) 为 TabHost 添加标签，然后新建一个 newTabSpec(new Tabspec)，并设置其标签、图标和内容。

(3) 分别设置 TabHost 的背景颜色、背景图片资源和当前显示哪一个标签。

(4) 定义标签切换事件处理方法 setOnTabChangedListener。

文件 Activity01.java 的具体实现代码如下所示：

```
public class Activity01 extends TabActivity
{
    //声明 TabHost 对象
    TabHost mTabHost;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //取得 TabHost 对象
        mTabHost = getTabHost();

        /* 为 TabHost 添加标签 */
        //新建一个 newTabSpec(newTabSpec)
        //设置其标签和图标(setIndicator)
        //设置内容(setContent)
       mTabHost.addTab(mTabHost.newTabSpec("test1")
            .setIndicator("TAB 1",getResources().getDrawable(R.drawable.img1))
            .setContent(R.id.textview1));
       mTabHost.addTab(mTabHost.newTabSpec("test2")
            .setIndicator("TAB 2",getResources().getDrawable(R.drawable.img2))
            .setContent(R.id.textview2));
       mTabHost.addTab(mTabHost.newTabSpec("test3")
            .setIndicator("TAB 3",getResources().getDrawable(R.drawable.img3))
            .setContent(R.id.textview3));

       //设置 TabHost 的背景颜色
       mTabHost.setBackgroundColor(Color.argb(150, 22, 70, 150));
       //设置 TabHost 的背景图片资源
       //mTabHost.setBackgroundResource(R.drawable.bg0);
```

```
        //设置当前显示哪一个标签
        mTabHost.setCurrentTab(0);

        //标签切换事件处理，setOnTabChangedListener
        mTabHost.setOnTabChangedListener(new OnTabChangeListener()
        {
         // TODO Auto-generated method stub
           @Override
           public void onTabChanged(String tabId)
           {
             Dialog dialog = new AlertDialog.Builder(Activity01.this)
                     .setTitle("善意的提醒")
                     .setMessage("现在选中了："+tabId+"标签")
                     .setPositiveButton("确定",
                     new DialogInterface.OnClickListener()
                     {
                         public void onClick(DialogInterface dialog, int whichButton)
                         {
                             dialog.cancel();
                         }
                     }).create();//创建按钮
             dialog.show();
           }
        });
    }
}
```

至此，整个演练就完美结束了。执行后的效果如图 5-26 所示，单击某个图片后会相应单击事件处理程序，显示对应的对话框。例如，单击第一个图片后，显示的对话框效果如图 5-27 所示。

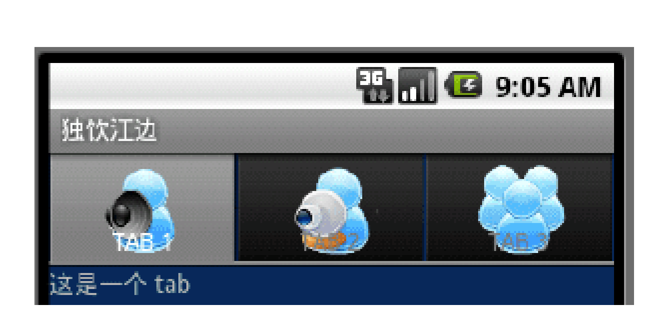


图 5-26　执行效果



图 5-27　显示的对话框效果

5.4　我的朋友 menu

因为是新人的缘故，初涉江湖的我几乎得不到什么任务，得不到我决定利用这段清闲的时间继续修炼秘籍，现在开始学习新的心法——menu 控件。

5.4.1　很友好的 menu

menu 控件的功能是为用户提供一个友好界面的显示效果。大部分的应用程序都包括两种人机互动方式，一种是直接通过 GUI 的 View，它可以满足大部分的交互操作；另外一种是应用 menu，当按下 menu 按钮后，会弹出与当前活动状态下应用程序相匹配的菜单。这两种方式相比较各有优势，而且可以很好地相辅相成。

Android 提供了以下三种菜单类型。

(1)　options menu：通过按 home 键来显示。

(2)　context menu：需要在 View 上按住 2s 后显示，context menu 是跟某个具体 View 绑定在一起的，在 Activity 中用 registerForContextMenu 为某个 View 注册 context menu，context menu 在显示前都会调用 onCreateContextMenu 来生成 menu，onContextItemSelected 用来处理选中的菜单项。

上述两种菜单类型是最常用的，这两种都有可以加入的子菜单，但是子菜单不能嵌套子菜单。

(3)　sub menu：不是很常用。

Android 可以对菜单项进行分组，能够把相似功能的菜单项分成同一个组，这样就可以通过调用 setGroupCheckable、setGroupEnabled 和 setGroupVisible 来设置菜单属性，而无须单独设置。

5.4.2　演练 menu

题　目	目　的	源码路径
演练 7	实战演练 menu 控件的用法	“光盘:\daima\5\menu”文件夹

第 1 步：新建工程文件后，先编写 main.xml 主文件，此文件是一个布局文件。具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:orientation="vertical" Android:layout_width="fill_parent"
    Android:layout_height="fill_parent">
    <TextView Android:layout_width="fill_parent"
        Android:layout_height="wrap_content" Android:text="@string/hello" />
    <Button Android:id="@+id/button1"
        Android:layout_width="100px"
        Android:layout_height="wrap_content" Android:text="@string/button1" />
    <Button Android:id="@+id/button2"
        Android:layout width="wrap content"
        Android:layout height="wrap content" Android:text="@string/button2" />
</LinearLayout>
```

通过上述代码，插入了一个 TextView 控件和两个 Button 控件，其中 TextView 用于显示文本。然后用“layout_width”设置 Button 的宽度，用“layout_height”设置 Button 的高度。最后，通过符号@来设置读取变量值并进行替换，具体说明如下。

- Android:text="@string/button1"：相当于<string name="button1">button1</string>。

- Android:text="@string/button2"：相当于<string name="button2">button2</string>。

上面的符号@用于提示 XML 文件的解析器要对@后面的名字进行解析。例如，上面的"@string/button1"，解析器会从 values/string.xml 中读取 Button1 这个变量值。

文件 string.xml 中定义了 TextView 和 Button 的值，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, ActivityMenu</string>
    <string name="app_name">HelloMenu</string>
    <string name="button1">button1</string>
    <string name="button2">button2</string>
</resources>
```

第 2 步：编写 Java 文件 ActivityMenu.java，其具体实现流程如下。

(1) 定义函数 onCreate，用于显示出 main.xml 描述的 Layout，并设置两个 Button 为不可见状态。

(2) 定义函数 onCreateOptionsMenu，用于生成 menu，此函数是一个回调方法，只有当按下手机设备上的 menu 按钮后，Android 才会生成一个包含两个子项的菜单。在具体实现流程上，将首先得到 super 函数调用后的返回值，并在 onCreateOptionsMenu()方法的最后返回；然后调用 menu.add()方法给 menu 添加一个项。

(3) 定义函数 onOptionsItemSelected，此函数是一个回调方法，只有当按下手机设备上的 menu 按钮后，Android 才会调用执行。而这个事件就是单击菜单里的某一项，即 MenuItem。

文件 ActivityMenu.java 的主要代码如下所示：

```
public class ActivityMenu extends Activity {
    public static final int ITEM0 = Menu.FIRST;
    public static final int ITEM1 = Menu.FIRST + 1;
    Button button1;
    Button button2;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
         button1 = (Button) findViewById(R.id.button1);
         button2 = (Button) findViewById(R.id.button2);
         /* 设置两个 button 不可见 */
         button1.setVisibility(View.INVISIBLE);
         button2.setVisibility(View.INVISIBLE);
             }
    @Override
    /*
     * menu.findItem(EXIT_ID);找到特定的 MenuItem
     * MenuItem.setIcon.可以设置 menu 按钮的背景
     */
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        menu.add(0, ITEM0, 0, "button1");
        menu.add(0, ITEM1, 0, "button2");
        menu.findItem(ITEM1);
```

```
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case ITEM0:
            actionClickMenuItem1();
        break;
        case ITEM1:
            actionClickMenuItem2(); break;
        }
        return super.onOptionsItemSelected(item);}
    /*
     * 点击第一个 menu 的第一个按钮执行的动作
     */
    private void actionClickMenuItem1(){
        setTitle("button1 可见");
        button1.setVisibility(View.VISIBLE);
        button2.setVisibility(View.INVISIBLE);
    }
    /*
     * 点击第二个 menu 的第一个按钮执行的动作
     */
    private void actionClickMenuItem2(){
        setTitle("button2 可见");
        button1.setVisibility(View.INVISIBLE);
        button2.setVisibility(View.VISIBLE);
    }
}
```

至此，整个演练就完美结束了，执行后的初始效果如图 5-28 所示；当点击设备上的 menu 键后会触发程序，并在屏幕中显示预先设置的已经隐藏的两个按钮，如图 5-29 所示。

图 5-28　初始效果



图 5-29　触发设备后的效果

5.5　Intent 和 Activity 深情相拥

我一直有一个疑问：前面的演练都是基于一个 Activity 的，那么可不可以一次使用多个

Activity，并且这些 Activity 之间可以相互跳转、相互传递数据呢？在秘籍中我找到了答案，可以实现！

5.5.1 Intent 调用另一个 Activity

秘籍上说：使用一个 Intent 调用另一个 Activity 的方法十分简单，只需要通过一段简短的代码即可实现，具体实现过程如下。

第 1 步：新建工程，在 string.xml 中添加两个字符串。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, Ex9_UI!</string>
    <string name="app_name">Ex9_UI</string>
    <string name="act1">This is Activity 1!</string>
    <string name="act2">This is Activity 2!</string>
</resources>
```

第 2 步：新建 color.xml 存放颜色值。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="black">#000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

第 3 步：修改 main.xml 布局，添加一个 TextView 和一个 Button。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    Android:layout_width="fill_parent"
    Android:layout_height="fill_parent"
    Android:background="@color/black"
    xmlns:Android="http://schemas.Android.com/apk/res/Android"
>
<TextView
    Android:id="@+id/text1"
    Android:textSize="24sp"
    Android:layout_width="186px"
    Android:layout_height="29px"
    Android:layout_x="70px"
    Android:layout_y="32px"
    Android:text="@string/act1"
></TextView>
<Button
    Android:id="@+id/button1"
    Android:layout_width="118px"
    Android:layout_height="wrap_content"
    Android:layout_x="100px"
    Android:layout_y="82px"
```

```
    Android:text="Go to Activity2"
></Button>
</AbsoluteLayout>
```

第 4 步：新建一个 secondlayout.xml 布局，并添加一个 TextView 和一个 Button。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    Android:layout_width="fill_parent"
    Android:layout_height="fill_parent"
    Android:background="@color/white"
    xmlns:Android="http://schemas.Android.com/apk/res/Android"
>
<TextView
    Android:id="@+id/text2"
    Android:textSize="24sp"
    Android:layout_width="186px"
    Android:layout_height="29px"
    Android:layout_x="70px"
    Android:layout_y="32px"
    Android:textColor="@color/black"
    Android:text="@string/act2"
></TextView>
<Button
    Android:id="@+id/button2"
    Android:layout_width="118px"
    Android:layout_height="wrap_content"
    Android:layout_x="100px"
    Android:layout_y="82px"
    Android:text="Go to Activity1"
></Button>
</AbsoluteLayout>
```

第 5 步：新建 SecondActivity.java 文件，然后添加内容。其主要代码如下：

```
public class SecondActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /* 载入 mylayout.xml Layout */
        setContentView(R.layout.mylayout);
        /* 以 findViewById()取得 Button 对象，并添加 onClickListener */
        Button b2 = (Button) findViewById(R.id.button2);
        b2.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                /* new 一个 Intent 对象，并指定要启动的 class */
```

```
                Intent intent = new Intent();
                intent.setClass(SecondActivity.this, Ex9_UI.class);
                /* 调用一个新的 Activity */
                startActivity(intent);
                /* 关闭原本的 Activity */
                SecondActivity.this.finish();
            }
        });
    }
}
```

第 6 步：修改 mainActivity.java 并添加对应的处理代码。其主要代码如下：

```
public class Ex9_UI extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /* 载入 main.xml Layout */
        setContentView(R.layout.main);
        /* 以 findViewById()取得 Button 对象，并添加 onClickListener */
        Button b1 = (Button) findViewById(R.id.button1);
        b1.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                /* new 一个 Intent 对象，并指定要启动的 class */
                Intent intent = new Intent();
                intent.setClass(Ex9_UI.this, SecondActivity.class);
                /* 调用一个新的 Activity */
                startActivity(intent);
                /* 关闭原本的 Activity */
                Ex9_UI.this.finish();
            }
        });
    }
}
```

第 7 步：在 AndroidManifest.xml 文件中添加 SecondActivity。其主要代码如下：

```
    <application Android:icon="@drawable/icon" Android:label="@string/app_name">
        <activity Android:name=".Ex9_UI"
                Android:label="@string/app_name">
            <intent-filter>
                <action Android:name="Android.intent.action.MAIN" />
                <category Android:name="Android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    <activity Android:name="SecondActivity"></activity>
```

这样，就简单地通过使用 Intent 调用了另一个 Activity。执行后会显示初始界面，单击按钮后会跳到另一个界面。秘籍中讲解得很详细，一看便可以明白。

5.5.2　演练 Intent 和 Activity 的合作

题　目	目　的	源码路径
演练 8	实战演练 Intent 和 Activity 联合使用的用法	“光盘：\daima\5\activity and intent”文件夹

第 1 步：创建工程文件后，先编写 main.xml 主文件。具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:orientation="vertical" Android:layout_width="fill_parent"
    Android:layout_height="fill_parent">
    <Button Android:id="@+id/button1"
        Android:layout width="wrap content"
        Android:layout_height="wrap_content" Android:text="button1" />
    <Button Android:id="@+id/button2"
        Android:layout_width="wrap_content"
        Android:layout_height="wrap_content" Android:text="button2" />
</LinearLayout>
```

通过上述代码，插入了两个 Button 控件。

第 2 步：编写主程序 ActivityMain.java，下面开始讲解主程序 ActivityMain.java 的具体实现过程。

(1)　首先创建 onCreat 函数，具体代码如下所示：

```
setContentView(R.layout.main);
        button1 = (Button) findViewById(R.id.button1);
        button1.setOnClickListener(listener1);
        button2 = (Button) findViewById(R.id.button2);
        button2.setOnClickListener(listener2);
        setTitle("ActivityMain");
```

在上述代码中，首先把 layout 目录中 main.xml 的 layout 显示出来，然后才能得到 button 的应用，依次绑定一个，单击监听器 OnClickListener。

(2)　编写第一个跳转代码处理程序，具体代码如下：

```
listener2 = new OnClickListener() {
            public void onClick(View v) {
                setTitle("当前 ActivityMain");
                Intent intent2 = new Intent(ActivityMain.this, Activity2.class);
                startActivity(intent2);
                ……
```

在上述代码中，实现了当单击 button2 按钮时，程序跳转到 Activity2。即在单击 button2 按钮时，系统反向调用绑定在 button2 上的监听器的 onClick 方法。

(3)　编写第二个跳转代码处理程序，具体代码如下：

```
listener1 = new OnClickListener() {
            public void onClick(View v) {
                Intent intent1 = new Intent(ActivityMain.this, Activity1.class);
                intent1.putExtra("activityMain", "from activityMain");
```

```
                startActivityForResult(intent1, REQUEST CODE);
                ...
```

在上述代码中，实现了当单击 button1 按钮时，程序从 ActivityMain 跳转到 Activity1。

(4) 编写函数 onActivityResult，此函数用于启动 Intent，并在新的 Activity 运行完毕后，执行原来 Activity 中的回调函数，此回调函数是 onActivityResult。对应代码如下所示：

```
@Override
   protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
        if (requestCode == REQUEST_CODE) {
            if (resultCode == RESULT_CANCELED)
                setTitle("取消");
            else if (resultCode == RESULT OK) {
                String temp=null;
                 Bundle extras = data.getExtras();
                   if (extras != null) {
                    temp = extras.getString("store");
                   }
                setTitle(temp);
            }
        }
    }
```

第 3 步：编写文件 Activity2.java，使用一个新的 Activity——Activity2，并实现它和 activity2.xml 的关联。具体代码如下所示：

```
public class Activity2 extends Activity {
    OnClickListener listener = null;
    Button button;
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity2);
       listener = new OnClickListener() {
            public void onClick(View v) {
                finish();
            }
        };
        button = (Button) findViewById(R.id.button4);
        button.setOnClickListener(listener);
        setTitle("在 Activity2");
    }
}
```

第 4 步：编写文件 Activity1.java，具体的实现流程如下。

(1) 单击 button1 按钮后 Activity 实现跳转，跳转到了 Activity1。在 Activity1 中，程序会把 ActivityMain 传递来的值显示在屏幕标题 title 中，具体实现代码如下：

```
Bundle extras = getIntent().getExtras();
           if (extras != null) {
               data = extras.getString("activityMain");
```

```
            }
        setTitle("在 Activity1:"+data);
```

(2) 单击 Activity1 中的 button3 按钮后，此 Activity 结束，此时会将需要返回的数据放置到 Intent 中，具体代码如下所示：

```
listener1 = new OnClickListener() {
            public void onClick(View v) {
                Bundle bundle = new Bundle();
                bundle.putString("store", "from Activity1");
                Intent mIntent = new Intent();
                mIntent.putExtras(bundle);
                setResult(RESULT_OK, mIntent);
                finish();
            }
        };
```

第 5 步：编写文件 AndroidManifest.xml，加入新的 Activity。具体代码如下所示：

```
<activity Android:name=".ActivityMain" Android:label="@string/app_name">
- <intent-filter>
  <action Android:name="Android.intent.action.MAIN" />
  <category Android:name="Android.intent.category.LAUNCHER" />
  </intent-filter>
  </activity>
  <!—下面是新加的 Activity-->
  <activity Android:name=".Activity1" />
  <activity Android:name=".Activity2" />
```

至此，整个演练就完美结束了，执行后的初始效果如图 5-30 所示；单击 button1 按钮后的效果如图 5-31 所示；单击 button3 按钮后的效果如图 5-32 所示；单击 button2 按钮后的效果如图 5-33 所示。

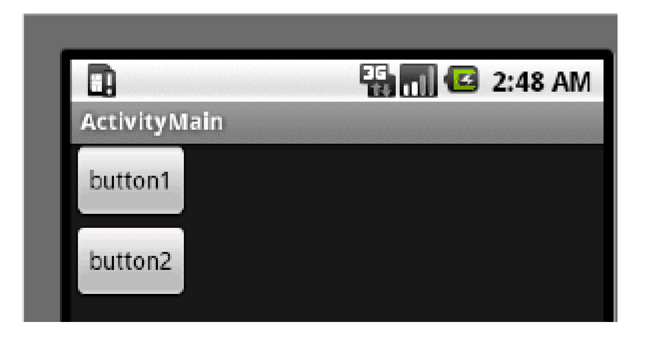


图 5-30　初始效果

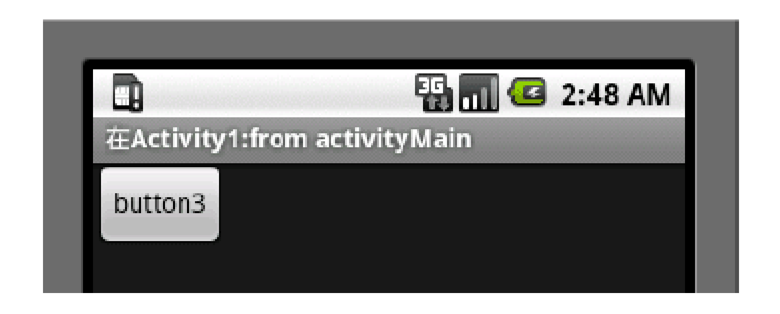


图 5-31　单击 button1 按钮后的效果

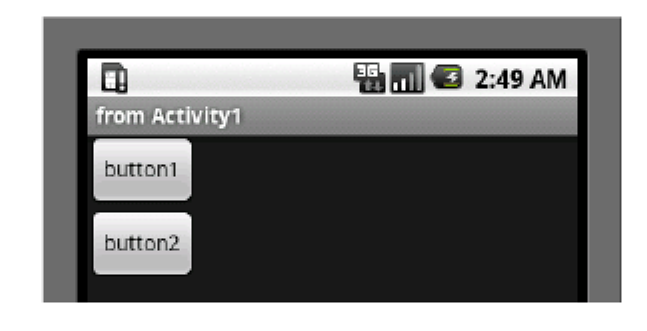


图 5-32　单击 button3 按钮后的效果

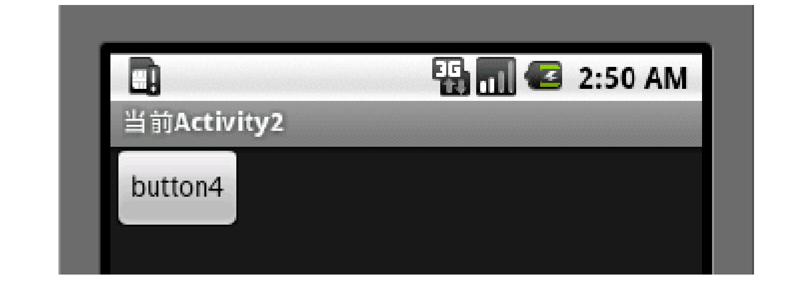


图 5-33　单击 button2 按钮后的效果

5.5.3　还可以重来

都说人生不能够重来，明天永远是新的一天。但是在 Android 中却可以通过 startActivityForResult

方法来实现将数据返回到前一个 Activity。秘籍中通过一段代码演示了重来的过程，具体流程如下。

第 1 步：修改 main.xml 布局，添加 UI 元素。

第 2 步：新建一个 mylayout.xml 布局，添加 UI 元素，具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
   Android:layout_width="fill_parent"
   Android:layout_height="fill_parent"
   xmlns:Android="http://schemas.Android.com/apk/res/Android"
>
<TextView
   Android:id="@+id/text1"
   Android:layout_width="wrap_content"
   Android:layout_height="wrap_content"
   Android:textSize="20sp"
   Android:layout_x="50px"
   Android:layout_y="72px"
></TextView>
<Button
   Android:id="@+id/button_back"
   Android:layout_width="100px"
   Android:layout_height="48px"
   Android:text="回上一页"
   Android:layout_x="110px"
   Android:layout_y="180px"
></Button></AbsoluteLayout>
```

第 3 步：新建一个 SecondActivity.java 的 Activity 子类。具体代码如下所示：

```
package zyf.Ex11_UI_A;
import Android.app.Activity;
import Android.os.Bundle;

public class BMIActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

第 4 步：在 AndroidManifest.xml 中添加 SecondActivity，具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:Android="http://schemas.Android.com/apk/res/Android"
     package="zyf.Ex11_UI_A"
     Android:versionCode="1"
     Android:versionName="1.0">
   <application Android:icon="@drawable/icon" Android:label="@string/app_name">
      <activity Android:name=".Ex11_UI_A"
              Android:label="@string/app_name">
```

```
        <intent-filter>
            <action Android:name="Android.intent.action.MAIN" />
            <category Android:name="Android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <!--下面必须在 AndroidManifest 中注册新的 Activity，否则程序出错-->
    <activity Android:name="BMIActivity"></activity>
</application>
    <uses-sdk Android:minSdkVersion="2" />
</manifest>
```

第 5 步：修改 mainActivity.java 代码。主要代码如下所示：

```
…
public class Ex11_UI_A extends Activity {
    protected int my_requestCode = 1550;
    private EditText edit_height;
    private RadioButton radiobutton_Man, radiobutton_Woman;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /* 载入 main.xml Layout */
        setContentView(R.layout.main);
        /* 以 findViewById()取得 Button 对象，并添加 onClickListener */
        Button ok = (Button) findViewById(R.id.button_OK);
        edit_height = (EditText) findViewById(R.id.height_Edit);
        radiobutton_Man = (RadioButton) findViewById(R.id.Sex_Man);
        radiobutton_Woman = (RadioButton) findViewById(R.id.Sex_Woman);
        ok.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                try {
                    /* 取得输入的身高 */
                    double height = Double.parseDouble(edit height.getText()
                            .toString());
                    /* 取得选择的性别 */
                    String sex = "";
                    if (radiobutton_Man.isChecked()) {
                        sex = "M";
                    } else {
                        sex = "F";
                    }
                    /* new 一个 Intent 对象，并指定 class */
                    Intent intent = new Intent();
                    intent.setClass(Ex11_UI_A.this, BMIActivity.class);
                    /* new 一个 Bundle 对象，并将要传递的数据传入 */
                    Bundle bundle = new Bundle();
                    bundle.putDouble("height", height);
                    bundle.putString("sex", sex);
                    /* 将 Bundle 对象 assign 给 Intent */
                    intent.putExtras(bundle);
                    /* 调用 Activity EX03_10_1 */
                    startActivityForResult(intent, my_requestCode);
```

```
                } catch (Exception e) {
                    // TODO: handle exception
                    Toast.makeText(Ex11_UI_A.this,
                          R.string.errorString, Toast.LENGTH_LONG).show();
                }
            }
        });
    }
    @Override
    /*新重写方法，等待返回结果*/
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        // TODO Auto-generated method stub
        super.onActivityResult(requestCode, resultCode, data);
        switch (resultCode) {
        case RESULT_OK:
            /* 取得来自 Activity2 的数据，并显示于画面上 */
            Bundle bunde = data.getExtras();
            String sex = bunde.getString("sex");
            double height = bunde.getDouble("height");
            edit_height.setText("" + height);
            if (sex.equals("M")) {
                radiobutton_Man.setChecked(true);
            } else {
                radiobutton_Woman.setChecked(true);
            }
            break;
        default:
            break;
        }
    }
}
```

第 6 步：修改 SecondActivity.java 的处理代码。主要代码如下所示：

```
public class BMIActivity extends Activity {
    private Intent intent;
    private Bundle bunde;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /* 加载 main.xml Layout */
        setContentView(R.layout.mylayout);
        /* 取得 Intent 中的 Bundle 对象 */
        intent = this.getIntent();
        bunde = intent.getExtras();
        /* 取得 Bundle 对象中的数据 */
        String sex = bunde.getString("sex");
        double height = bunde.getDouble("height");
        /* 判断性别 */
        String sexText = "";
        if (sex.equals("M")) {
            sexText = "男性";
```

```
        } else {
            sexText = "女性";
        }
        /* 取得标准体重 */
        String weight = this.getWeight(sex, height);
        /* 设置输出文字 */
        TextView tv1 = (TextView) findViewById(R.id.text1);
        tv1.setText("你是一位" + sexText + "\n 你的身高是" + height +
                            "厘米\n 你的标准体重是"+ weight + "公斤");
        /* 以 findViewById()取得 Button 对象，并添加 onClickListener */
        Button b1 = (Button) findViewById(R.id.button_back);
        b1.setOnClickListener(new Button.OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                /* 返回 result 回上一个 activity，即返回刚刚接收的 Intent */
                BMIActivity.this.setResult(RESULT_OK, intent);
                /* 结束这个 activity */
                BMIActivity.this.finish();
            }
        });
    }
    /* 四舍五入的 method */
    private String format(double num) {
        NumberFormat formatter = new DecimalFormat("0.00");
        String s = formatter.format(num);
        return s;
    }
    /* 以 findViewById()取得 Button 对象，并添加 onClickListener */
    private String getWeight(String sex, double height) {
        String weight = "";
        if (sex.equals("M")) {
            weight = format((height - 80) * 0.7);
        } else {
            weight = format((height - 70) * 0.6);
        }
        return weight;
    }
}
```

至此，整个代码就演示完毕了。执行后将首先显示一个信息输入表单，供用户选择性别并输入身高，单击“计算”按钮后将输出用户的标准体重，运行结果如图 5-34 所示。



图 5-34　运行结果

5.6 罗列有序的兵器库

一天的训练令我有一些疲惫，在洗澡之后感觉精神许多。我决定利用这难得的精神来继续修炼秘籍，今天的任务是学习 ListView 心法。Android 中的 ListView 能够展示一个友好的秩序，就像兵器库中摆放有序的兵刃一样；ListView 是 Android 开发中最常用的组件之一，能够在屏幕内实现列表显示；ListView 是通过一个 Adapter 来构建显示的，通常有三种 Adapter 可以使用，分别为：ArrayAdapter、SimpleAdapter 和 CursorAdapter。

5.6.1 ArrayAdapter 的基本用法

ArrayAdapter 可以接收一个数组，也可以将 List 作为参数，来构建数据并显示。例如，在下面的代码中，先创建 Test 继承 ListActivity，然后在里面传入一个 string 数组，具体代码如下：

```
public class ListTest extends ListActivity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      String[] sw = new String[100];
      for (int i = 0; i < 100; i++) {
         sw[i] = "listtest_" + i;
      }
      ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
      Android.R.layout.simple_list_item_1,sw);//使用系统已经实现好的 xml 文件
      simple_list_item_1
      setListAdapter(adapter);
   }
}
```

程序执行后的效果如图 5-35 所示。

从图 5-35 所示的执行效果可以看出，不需要加载自己的 Layout，只使用系统已经实现的 Layout 就能很快地实现 ListView 效果。

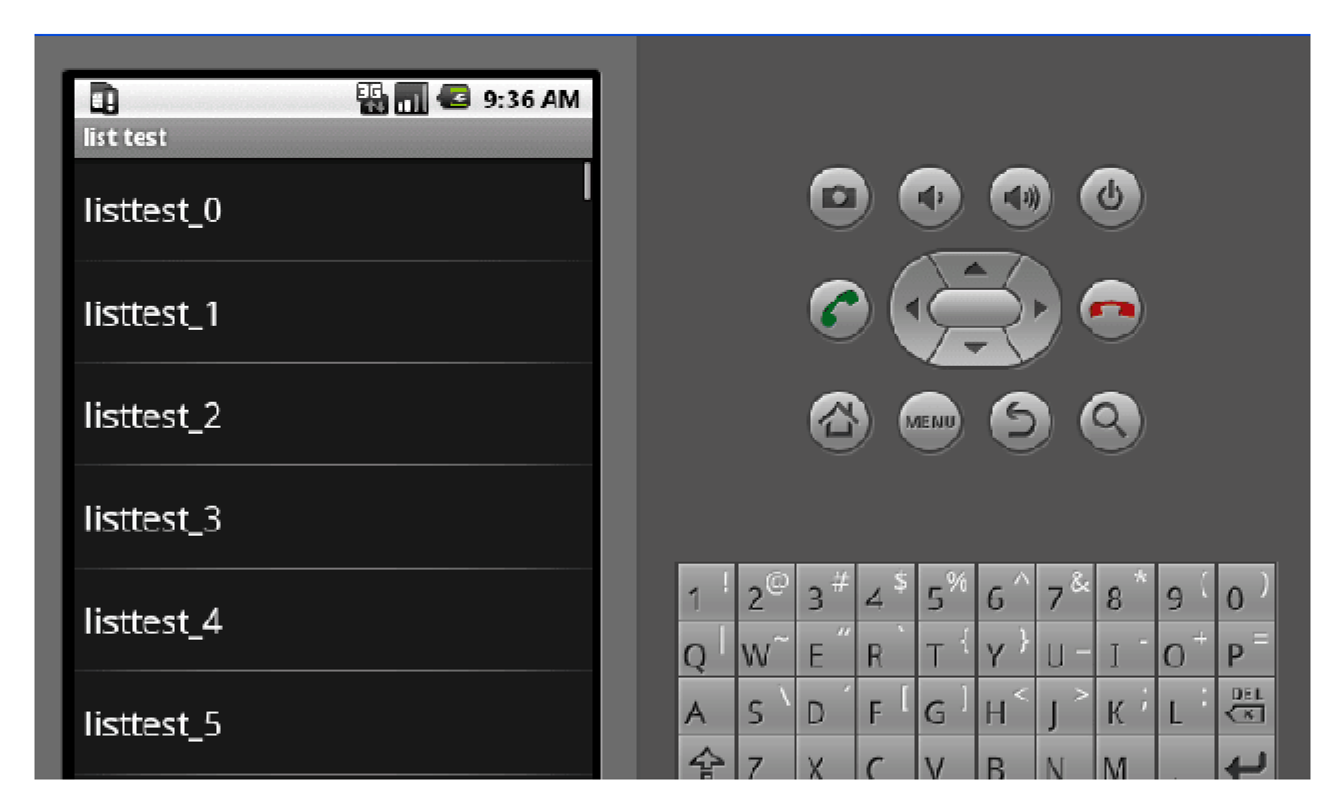


图 5-35 程序执行后的效果

5.6.2　使用 SimpleAdapter 实现列表样式

题　目	目　的	源码路径
演练 9	实战演练使用 SimpleAdapter 实现 ListView 的流程	“光盘：\daima\5\my_list”文件夹

第 1 步：构建 list 并设置每项有一个 map 图片，然后创建 TestList 类继承 Activity。具体代码如下所示：

```
super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ArrayList<HashMap<String, Object>> users = new ArrayList<HashMap<String,
        Object>>();
        for (int i = 0; i < 10; i++) {
            HashMap<String, Object> user = new HashMap<String, Object>();
            user.put("img", R.drawable.user);
            user.put("username", "名字(" + i+")");
            user.put("age", (11 + i) + "");
            users.add(user);
        }
        SimpleAdapter saImageItems = new SimpleAdapter(this,
                users,// 数据来源
                R.layout.user,//每一个 user xml 相当于 ListView 的一个组件
                new String[] { "img", "username", "age" },
                // 分别对应 view 的 id
                new int[] { R.id.img, R.id.name, R.id.age });
        // 获取 listview
((ListView) findViewById(R.id.users)).setAdapter(saImageItems);
```

第 2 步：编写文件 main.xml 实现布局，插入 3 个 TextView，其中，ListView 前面的是标题行，ListView 相当于用来显示数据的容器，里面每行是一个用户信息。

第 3 步：编写文件 use.xml，用于定义用户信息布局。具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
        Android:layout width="fill parent"
        xmlns:Android="http://schemas.Android.com/apk/res/Android"
        Android:layout_height="wrap_content"
        >
        <TableRow >
        <ImageView
             Android:layout_width="wrap_content"
             Android:layout_height="wrap_content"
             Android:id="@+id/img">
        </ImageView>
        <TextView
             Android:layout_height="wrap_content"
             Android:layout width="150px"
             Android:id="@+id/name">
        </TextView>
        <TextView
```

```
            Android:layout height="wrap content"
            Android:layout_width="170px"
            Android:id="@+id/age">
        </TextView>
        </TableRow>
</TableLayout>
```

这样就设置了一个每行包含一个 img 和两个文字信息的文件，这个文件以参数的形式通过 Adapter 在 ListView 中显示。

第 4 步：编写主处理文件 ListTest.java，通过 SimpleAdapter 来显示每块区域的用户信息。具体代码如下所示：

```
SimpleAdapter saImageItems = new SimpleAdapter(this,
             users,                  // 数据来源
             R.layout.user,          //每一个 user xml 相当于 ListView 的一个组件
             new String[] { "img", "username", "age" },
             // 分别对应 view 的 id
             new int[] { R.id.img, R.id.name, R.id.age });
```

至此，整个演练就完美地完成了，执行后的效果如图 5-36 所示。

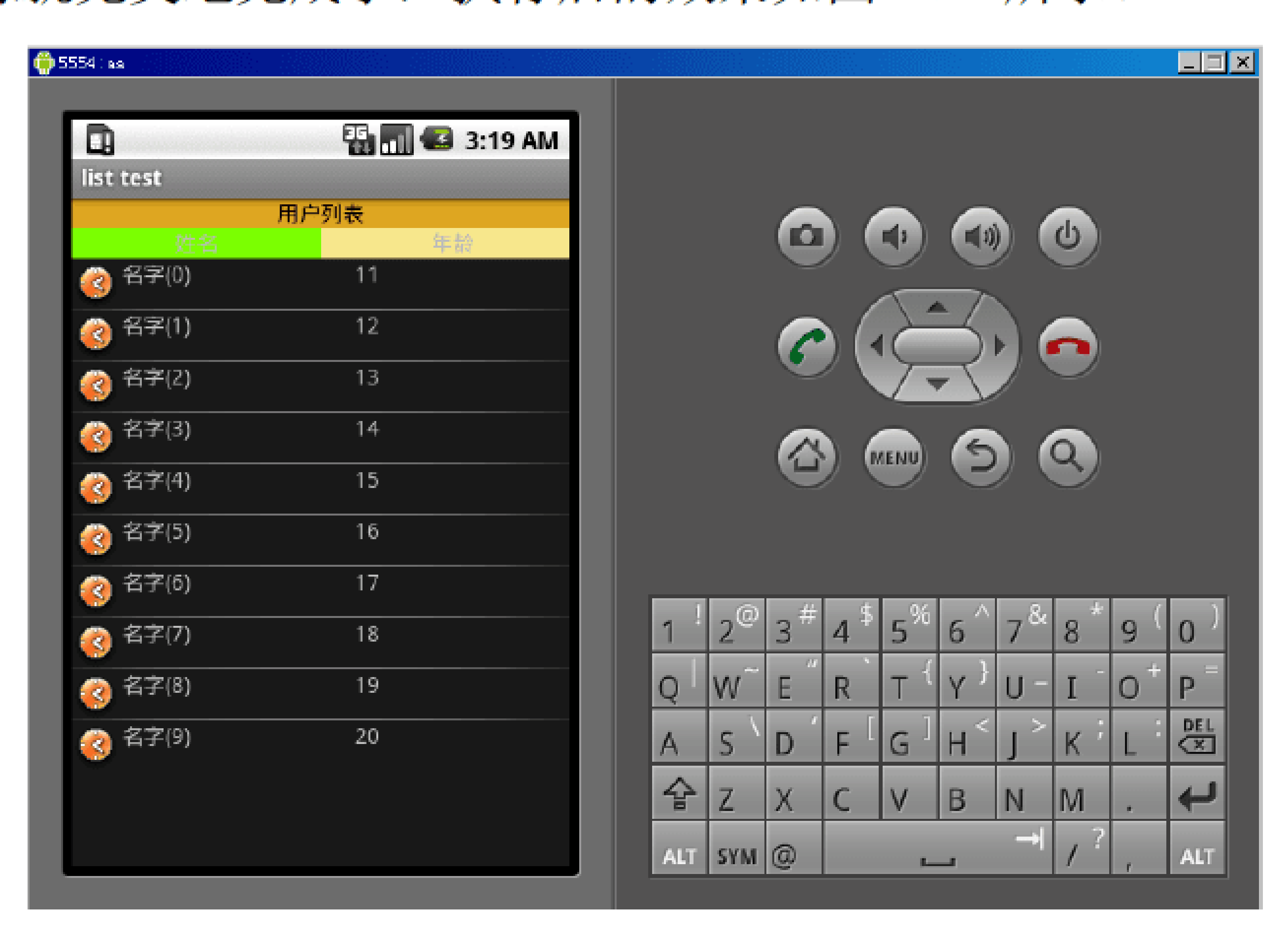


图 5-36　运行效果

5.7　使用对话框控件

秘籍中写道：交流很重要，在手机系统中交流也同样重要。在 Android 中，对话功能是通过 Dialog 控件实现的，其功能是在手机屏幕中实现互动对话框的效果。

题　目	目　的	源码路径
演练 10	实战演练使用 Dialog 控件的流程	“光盘：\daima\5\UseDialog”文件夹

第 1 步：新建 Android 工程后，编写主布局文件 main.xml。

第 2 步：开始编写程序文件 ActivityMain.java，具体实现过程如下。

(1)　首先看文件 ActivityMain.java 中的 OnCreater()方法。

- 方法 findViewById 通过组件的 ID 返回对这个组件的引用。
- 方法 setOnClickListener 为 button1 设置一个单击监听器。
- onClick 为单击 Button 后的回调函数。
- showDialog 是 Activity 中的函数，用于将 ID 为 Dialog1 的 Dialog 显示出来。

具体代码如下所示：

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.alert_dialog);
        Button button1 = (Button) findViewById(R.id.button1);
        button1.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                showDialog(DIALOG1);
            }
        });
        Button button2 = (Button) findViewById(R.id.button2);
        button2.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                showDialog(DIALOG2);
            }
        });
        Button button3 = (Button) findViewById(R.id.button3);
        button3.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                showDialog(DIALOG3);
            }
        });
        Button button4 = (Button) findViewById(R.id.button4);
        button4.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                showDialog(DIALOG4);
            }
        });
    }
```

(2) 方法 onCreateDialog 是一个回调函数，能够根据不同的 Dialog 的 ID 生成不同的 Dialog，例如 buildDialog1 函数用于生成第一个要显示的 Dialog，具体代码如下所示：

```
@Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case DIALOG1:
            return buildDialog1(ActivityMain.this);
        case DIALOG2:
            return buildDialog2(ActivityMain.this);
        case DIALOG3:
            return buildDialog3(ActivityMain.this);
        case DIALOG4:
            return buildDialog4(ActivityMain.this);
        }
        return null;
    }
```

(3) 实现函数 buildDialog1、buildDialog2、buildDialog3 和 buildDialog4，具体代码如下所示：

```
private Dialog buildDialog1(Context context) {
      /*创建一个 AlertDialog.Builder builder 对象*/
        AlertDialog.Builder builder = new AlertDialog.Builder(context);
      /*给 AlertDialog 预设一个图片*/
        builder.setIcon(R.drawable.alert_dialog_icon);
      /*给 AlertDialog 预设一个标题*/
        builder.setTitle(R.string.alert_dialog_two_buttons_title);
      /*设置按钮的属性*/
        builder.setPositiveButton(R.string.alert_dialog_ok,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                        setTitle("点击了对话框上的确定按钮");
                    }
                });
        builder.setNegativeButton(R.string.alert_dialog_cancel,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                        setTitle("点击了对话框上的取消按钮");
                    }
                });
        return builder.create();
    }
    private Dialog buildDialog2(Context context) {
        AlertDialog.Builder builder = new AlertDialog.Builder(context);
        builder.setIcon(R.drawable.alert_dialog_icon);
        builder.setTitle(R.string.alert dialog two buttons msg);
        builder.setMessage(R.string.alert_dialog_two_buttons2_msg);
        builder.setPositiveButton(R.string.alert_dialog_ok,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                        setTitle("点击了对话框上的确定按钮");
                    }
                });
        builder.setNeutralButton(R.string.alert_dialog_something,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                        setTitle("这是点击了对话框上的进入详细按钮");
                    }
                });
        builder.setNegativeButton(R.string.alert_dialog_cancel,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                        setTitle("点击了对话框上的取消按钮");
                    }
                });
        return builder.create();
    }
    private Dialog buildDialog3(Context context) {
        LayoutInflater inflater = LayoutInflater.from(this);
        final View textEntryView = inflater.inflate(
```

```
                R.layout.alert dialog text entry, null);
        AlertDialog.Builder builder = new AlertDialog.Builder(context);
        builder.setIcon(R.drawable.alert_dialog_icon);
        builder.setTitle(R.string.alert_dialog_text_entry);
        builder.setView(textEntryView);
        builder.setPositiveButton(R.string.alert_dialog_ok,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                        setTitle("点击了对话框上的确定按钮");
                    }
                });
        builder.setNegativeButton(R.string.alert dialog cancel,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                        setTitle("点击了对话框上的取消按钮");
                    }
                });
        return builder.create();
    }
        private Dialog buildDialog4(Context context) {
        ProgressDialog dialog = new ProgressDialog(context);
        dialog.setTitle("处理中");
        dialog.setMessage("请稍后……");
        return  dialog;
    }
}
```

上述 4 个函数的实现原理是一样的，具体说明如下。

- buildDialog1。

  onClick 方法是监听器中的回调方法，当单击 Dialog 按钮时，系统会回调这个方法。

  setNeutralButton 方法和 setPositiveButton 方法相对应，主要用于设置、取消按钮的一些属性。

  执行 builder.create 后，会生成一个配置好的 Dialog。

- buildDialog2。

  当单击第二个 button 后，会执行 buildDialog2，其中方法 setNeutralButton 用于设置中间按钮中的一些属性，具体设置的方法和 buildDialog1 中的一样。

- buildDialog3。

  当单击第三个 button 后，会执行 buildDialog3，其中通过 LayoutInflater 类的 inflater 方法，可以将一个 XML 布局变为一个 View 实例。下面的语句是整个 Dialog 的精髓：

```
builder.setView(textEntryView);
```

  通过上述方法可以将实现好的个性化的 View 放置到 Dialog 中去，此处的 textEntryView 和 alert_dialog_entry.xml 定义的布局相关联。

- buildDialog4。

  此程序最为简单，执行后将会显示一个等待界面。

第 3 步：编写文件 alert_dialog_text_entry.xml，其中，“Android:textAppearance="?Android:attr/textAppearanceMedium"”用于设置 TextView 里面文字的字体；“EditText”是一个文本输入框；

"Android:password="true""用于设置输入框是密码输入框，只显示星号；在"Android:capitalize="none""中，通过 capitalize 属性设置输入字符的大小写，none 表示首字母小写。

第 4 步：编写文件 alert_dialog.xml，用于设置各个按钮上显示的文本。

至此，整个演练就全部结束了。执行后的初始效果如图 5-37 所示；单击第一个 button 后的效果如图 5-38 所示；单击第二个 button 后的效果如图 5-39 所示；单击第三个 button 后的效果如图 5-40 所示；单击第四个 button 后的效果如图 5-41 所示。



图 5-37　初始效果



图 5-38　单击第一个 button 后的效果

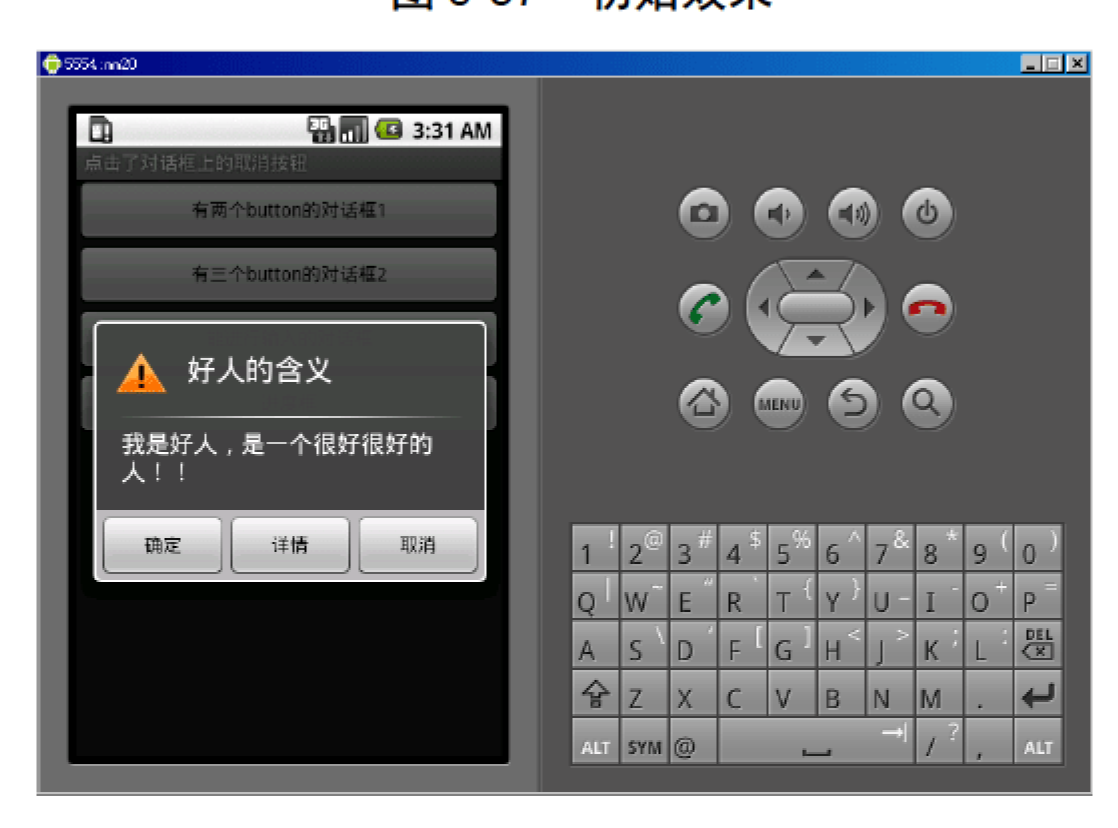


图 5-39　单击第二个 button 后的效果



图 5-40　单击第三个 button 后的效果



图 5-41　单击第四个 button 后的效果

5.8 一个善意的提醒

回家的路上，我仍不忘记练习心法。翻开秘籍，看到上面写道：生活中有提醒，Android 中也有提醒。在 Android 中，可以通过 Toast 和 Notification 控件来实现提醒功能。同 Dialog 相比，该类型的提醒更加友好和温馨，而且不会打断用户的当前操作。本节将详细讲解 Toast 和 Notification 控件的具体使用方法。

5.8.1 Toast 提醒你

Toast 是 Android 中用来显示信息的一种机制，与 Dialog 不同的是 Toast 没有焦点，并且 Toast 显示的时间有限，经过一定的时间后就会自动消失。

5.8.2 Notification 提醒你

看名字就知道，Notification 同提醒有关。通常 Notification 和 NotificationManager 一块使用，其主要功能如下。

1. NotificationManager 和 Notification 用来设置通知

通知设置的操作相对比较简单，基本的使用方式就是新建一个 Notification 对象，然后设置好通知的各项参数，再使用系统后台运行的 NotificationManager 服务将通知发出来。基本操作步骤如下。

(1) 创建 NotificationManager 对象 mNotificationManager，代码如下所示：

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager) getSystemService(ns);
```

(2) 创建一个新的 Notification 对象，代码如下所示：

```
Notification notification = new Notification();
notification.icon = R.drawable.notification_icon;
```

也可以使用稍微复杂一些的方式创建 Notification，代码如下所示：

```
int icon = R.drawable.notification_icon;    //通知图标
CharSequence tickerText = "Hello";          //状态栏(Status Bar)显示的通知文本提示
long when = System.currentTimeMillis();     //通知产生的时间，会在通知信息里显示
Notification notification = new Notification(icon, tickerText, when);
```

(3) 填充 Notification 的各个属性，代码如下所示：

```
Context context = getApplicationContext();
CharSequence contentTitle = "My notification";
CharSequence contentText = "Hello World!";
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
notification.setLatestEventInfo(context, contentTitle, contentText, contentIntent);
```

Notification 提供了如下几种手机提示方式.

- 在状态栏(Status Bar)显示通知文本提示，例如：

```
notification.tickerText = "hello";
```

- 发出提示音，例如：

```
notification.defaults |= Notification.DEFAULT SOUND;
notification.sound = Uri.parse("file:///sdcard/notification/ringer.mp3");
notification.sound  =  Uri.withAppendedPath(Audio.Media.INTERNAL_CONTENT_URI,
"6");
```

- 手机振动，例如：

```
notification.defaults |= Notification.DEFAULT_VIBRATE;
long[] vibrate = {0,100,200,300};
notification.vibrate = vibrate;
```

- LED 灯闪烁，例如：

```
notification.defaults |= Notification.DEFAULT_LIGHTS;
notification.ledARGB = 0xff00ff00;
notification.ledOnMS = 300;
notification.ledOffMS = 1000;
notification.flags |= Notification.FLAG_SHOW_LIGHTS;
```

(4) 发送通知，例如：

```
private static final int ID_NOTIFICATION = 1;
mNotificationManager.notify(ID_NOTIFICATION, notification);
```

2. 更新通知

如果需要更新一个通知，只需要在设置好 Notification 之后，再调用 setLatestEventInfo，然后重新发送一次通知即可。

为了更新一个已经触发过的 Notification，传入相同的 ID。既可以传入相同的 Notification 对象，又可以是一个全新的对象。只要 ID 相同，新的 Notification 对象就会替换状态条图标和扩展的状态窗口的细节。

另外，还可以使用 ID 来取消 Notification，具体取消功能是通过调用 NotificationManager 的 cancel 方法实现的，例如：

```
notificationManager.cancel(notificationRef);
```

当取消一个 Notification 时，会移除它的状态条图标以及清除在扩展的状态窗口中的信息。

5.8.3 演练 Toast 和 Notification

题　目	目　的	源码路径
演练11	实战演练使用 Toast 和 Notification 实现提醒功能	“光盘:\daima\5\toast and notification”文件夹

第 1 步：新建一个 Android 工程文件，然后编写 main.xml 布局文件。具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:orientation="vertical" Android:layout_width="fill_parent"
    Android:layout_height="fill_parent">
    <Button Android:id="@+id/button1"
        Android:layout_width="wrap_content"
        Android:layout_height="wrap_content" Android:text="介绍 Notification" />
    <Button Android:id="@+id/button2"
        Android:layout_width="wrap_content"
        Android:layout_height="wrap_content" Android:text="介绍 Toast" />
</LinearLayout>
```

通过上述代码插入了两个 Button 按钮，执行后的效果如图 5-42 所示。

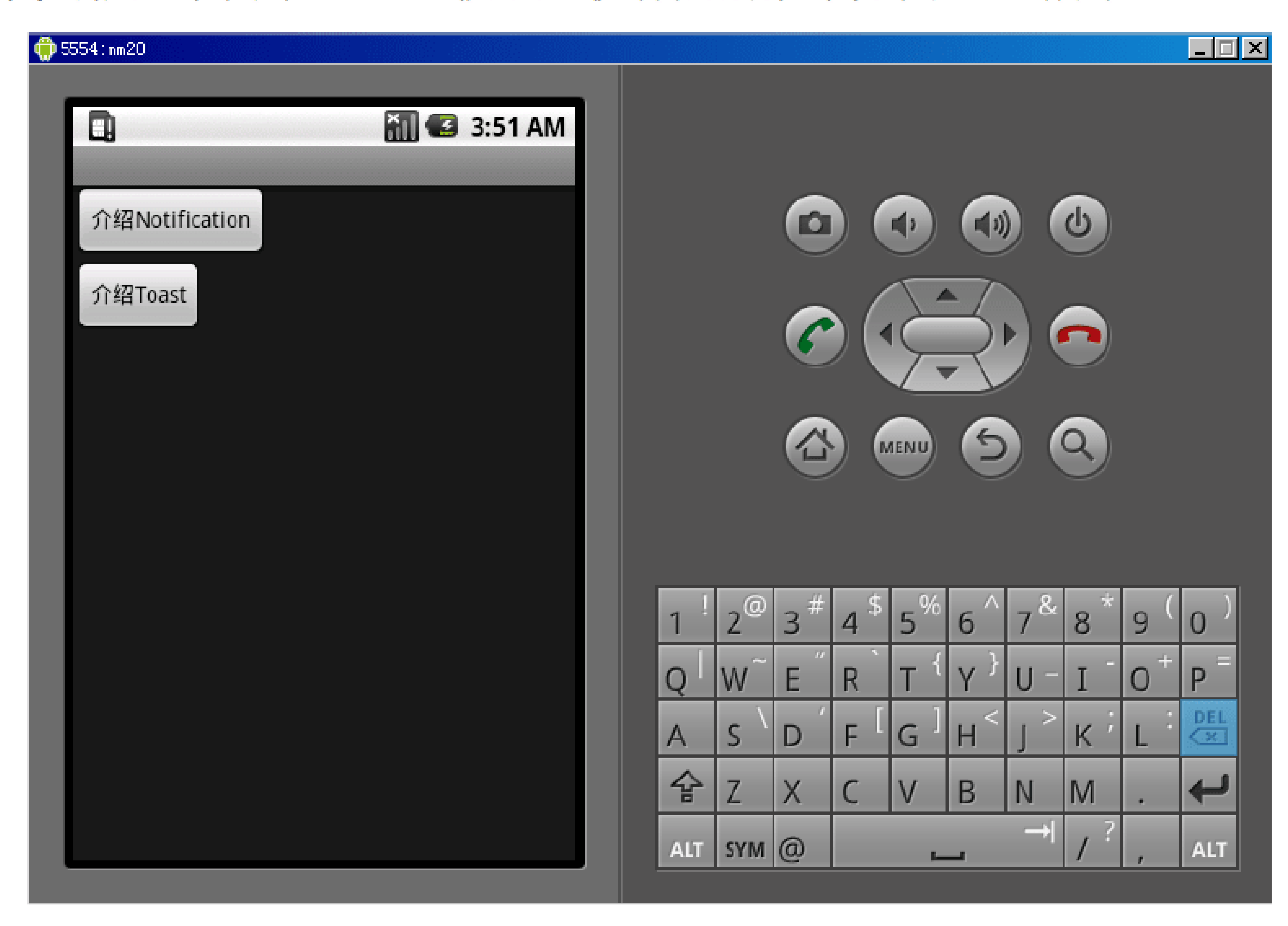


图 5-42　插入两个 Button 按钮

第 2 步：编写处理文件 ActivityMain.java。具体代码如下所示：

```
public class ActivityMain extends Activity {
    OnClickListener listener1 = null;
    OnClickListener listener2 = null;
    Button button1;
    Button button2;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        listener1 = new OnClickListener() {
            public void onClick(View v) {
                setTitle("讲解 Notification");
                Intent intent = new Intent(ActivityMain.this,
                        ActivityMainNotification.class);
                startActivity(intent);
            }
        };
        listener2 = new OnClickListener() {
            public void onClick(View v) {
```

```
                setTitle("讲解 Toast");
                Intent intent = new Intent(ActivityMain.this,
                        ActivityToast.class);
                startActivity(intent);
            }
        };
        setContentView(R.layout.main);
        button1 = (Button) findViewById(R.id.button1);
        button1.setOnClickListener(listener1);
        button2 = (Button) findViewById(R.id.button2);
        button2.setOnClickListener(listener2);
    }
}
```

在上述代码中，对两个 Button 绑定了单击监听器 OnClickListener，当单击这两个 Button 时，会跳转到新的 Activity。

第 3 步：编写第一个 Button 的处理程序，即单击图 5-42 中的“介绍 Notification”按钮后，执行 ActivityMainNotification.java。其主要代码如下所示：

```
public class ActivityMainNotification extends Activity {
    private static int NOTIFICATIONS_ID = R.layout.activity_notification;
    private NotificationManager mNotificationManager;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_notification);
        Button button;
        mNotificationManager = (NotificationManager) getSystemService
        (NOTIFICATION_SERVICE);
        button = (Button) findViewById(R.id.sun_1);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setWeather("好", "天气", "好", R.drawable.sun);
            }
        });
        button = (Button) findViewById(R.id.cloudy_1);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setWeather("一般", "天气", "一般", R.drawable.cloudy);
            }
        });
        button = (Button) findViewById(R.id.rain_1);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setWeather("不好", "天气", "不好", R.drawable.rain);
            }
        });
        button = (Button) findViewById(R.id.defaultSound);
        button.setOnClickListener(new Button.OnClickListener() {
```

```
            public void onClick(View v) {
                setDefault(Notification.DEFAULT_SOUND);
            }
        });
        button = (Button) findViewById(R.id.defaultVibrate);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setDefault(Notification.DEFAULT_VIBRATE);
            }
        });
        button = (Button) findViewById(R.id.defaultAll);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setDefault(Notification.DEFAULT_ALL);
            }
        });
        button = (Button) findViewById(R.id.clear);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                mNotificationManager.cancel(NOTIFICATIONS_ID);
            }
        });
    }
    private void setWeather(String tickerText, String title, String content,
            int drawable) {
        Notification notification = new Notification(drawable, tickerText,
                System.currentTimeMillis());
        PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
                new Intent(this, ActivityMain.class), 0);
        notification.setLatestEventInfo(this, title, content, contentIntent);
        mNotificationManager.notify(NOTIFICATIONS_ID, notification);
    }
    private void setDefault(int defaults) {
        PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
                new Intent(this, ActivityMain.class), 0);
        String title = "天气预报";
        String content = "晴";
        final Notification notification = new Notification(R.drawable.sun,
                content, System.currentTimeMillis());
        notification.setLatestEventInfo(this, title, content, contentIntent);
        notification.defaults = defaults;
        mNotificationManager.notify(NOTIFICATIONS_ID, notification);
    }
}
```

我感觉上述代码有一点复杂，于是将秘籍拿出来好好研究一番，一炷香的时间过后我豁然开朗了。

(1) 因为所有的 Notification 都是通过 NotificationManager 来管理的，所以应该首先得到

NotificationManager 实例，以便管理这个 Activity 中的通知信息：

```
mNotificationManager=(NotificationManager)getSystemService(NOTIFICATION_SERV
ICE);
```

(2) 函数 setWeather 是 ActivityMainNotification 中的重要函数之一，它实例化了一个 Notification，并将这个 Notification 显示出来。

(3) 在下面的代码中包含了 3 个参数，具体说明如下。

```
Notification notification = new Notification(drawable, tickerText,
                System.currentTimeMillis());
```

- ❑ 第 1 个：要显示图片的 ID。
- ❑ 第 2 个：显示的文本文字。
- ❑ 第 3 个：Notification 显示的时间，一般是立即显示，时间就是 System.currentTimeMillis()。

(4) 函数 setDefault：它是 ActivityMainNotification.java 中的一个重要函数，在此函数中初始化了一个 Notification，在设置 Notification 时使用了其默认值的形式，即：

```
notification.defaults = defaults;
```

另外，在上述程序中还用到了以下几种表现形式。

- ❑ Notification.DEFAULT_VIBRATE：表示当前的 Notification 显示出来时手机会发出震动。
- ❑ Notification.DEFAULT_SOUND：表示当前的 Notification 显示出来时手机会伴随音乐铃声。
- ❑ Notification.DEFAULT_ALL：表示当前的 Notification 显示出来时手机既会震动，也会伴随音乐。

这样当单击第一个 Button 后会执行上述处理程序，来到对应的新界面，新界面的布局文件是由 Activity_notification.xml 实现的，其主要代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:Android="http://schemas.Android.com/apk/res/Android"
   Android:layout_width="fill_parent"
   Android:layout_height="fill_parent">
      <LinearLayout
      Android:orientation="vertical"
      Android:layout_width="fill_parent"
      Android:layout_height="wrap_content">
              <LinearLayout
         Android:orientation="vertical"
         Android:layout_width="fill_parent"
         Android:layout_height="wrap_content">
               <Button
            Android:id="@+id/sun_1"
            Android:layout_width="wrap_content"
            Android:layout_height="wrap_content"
            Android:text="适合" />
               <Button
            Android:id="@+id/cloudy_1"
```

```
            Android:layout_width="wrap_content"
            Android:layout_height="wrap_content"
            Android:text="不太适合" />

          <Button
            Android:id="@+id/rain_1"
            Android:layout_width="wrap_content"
            Android:layout_height="wrap_content"
            Android:text="一点也不适合" />
    </LinearLayout>
            <TextView
        Android:layout_width="wrap_content"
        Android:layout_height="wrap_content"
        Android:layout_marginTop="20dip"
        Android:text="高级提示" />
    <LinearLayout
        Android:orientation="vertical"
        Android:layout_width="fill_parent"
        Android:layout_height="wrap_content">
               <Button
            Android:id="@+id/defaultSound"
            Android:layout_width="wrap_content"
            Android:layout_height="wrap_content"
            Android:text="有声音的提示" />
                <Button
            Android:id="@+id/defaultVibrate"
            Android:layout_width="wrap_content"
            Android:layout_height="wrap_content"
            Android:text="振动的提示" />
                         <Button
            Android:id="@+id/defaultAll"
            Android:layout_width="wrap_content"
            Android:layout_height="wrap_content"
            Android:text="声音+振动的提示" />

    </LinearLayout>
            <Button Android:id="@+id/clear"
        Android:layout_width="wrap_content"
        Android:layout_height="wrap_content"
        Android:layout_marginTop="20dip"
        Android:text="清除提示" />
            </LinearLayout>
</ScrollView>
```

上述代码执行后，新界面的效果如图 5-43 所示。

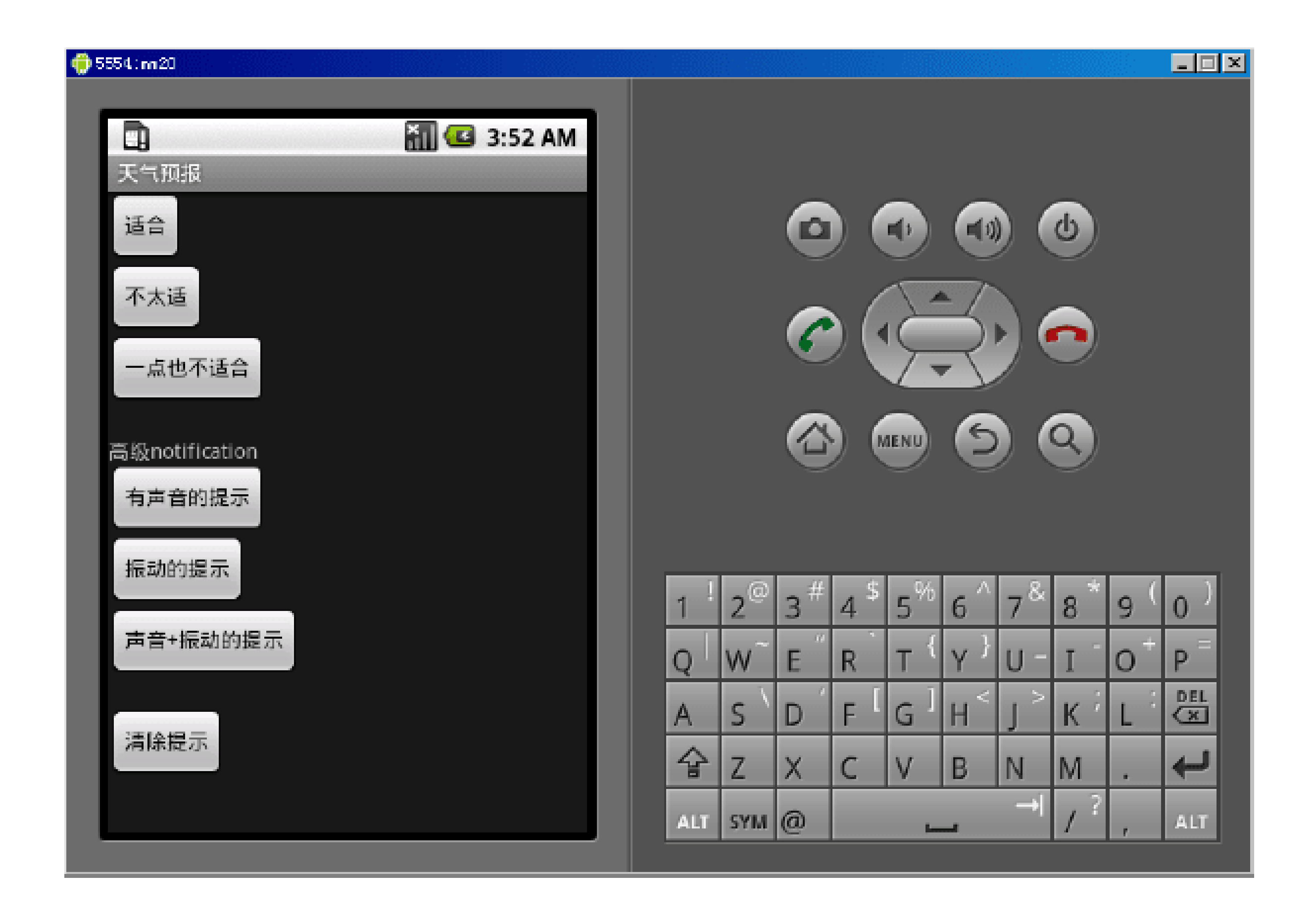


图 5-43　执行后的效果

当单击图 5-43 中的 Button 后，会根据上述处理文件实现某种效果，例如，单击“有声音的提示”按钮后，会发出声音。

第 4 步：编写第二个 Button 的处理程序，即单击图 5-42 中的“介绍 Toast”按钮后，执行 ActivityToast.java。其主要代码如下所示：

```
public class ActivityToast extends Activity {

    OnClickListener listener1 = null;
    OnClickListener listener2 = null;
    Button button1;
    Button button2;
    private static int NOTIFICATIONS_ID = R.layout.activity_toast;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        listener1 = new OnClickListener() {
            public void onClick(View v) {
                setTitle("短时显示 Toast");
                showToast(Toast.LENGTH_SHORT);
            }
        };
        listener2 = new OnClickListener() {
            public void onClick(View v) {
                setTitle("长时显示 Toast");
                showToast(Toast.LENGTH_LONG);
                showNotification();
            }
        };
        setContentView(R.layout.activity_toast);
        button1 = (Button) findViewById(R.id.button1);
        button1.setOnClickListener(listener1);
        button2 = (Button) findViewById(R.id.button2);
        button2.setOnClickListener(listener2);
    }
    protected void showToast(int type) {
```

```
        View view = inflateView(R.layout.toast);
        TextView tv = (TextView) view.findViewById(R.id.content);
        tv.setText("欢迎加入爬山摄影户外俱乐部，强身健体、增进友谊，共创和谐社会！");
      /*实例化 Toast*/
        Toast toast = new Toast(this);
        toast.setView(view);
        toast.setDuration(type);
        toast.show();
    }

    private View inflateView(int resource) {
        LayoutInflater vi = (LayoutInflater) getSystemService(Context.LAYOUT
        INFLATER SERVICE);
        return vi.inflate(resource, null);
    }
    protected void showNotification() {
        NotificationManager notificationManager = (NotificationManager)
        getSystemService(NOTIFICATION_SERVICE);
        CharSequence title = "最专业的户外拓展俱乐部";
        CharSequence contents = "户外合作论坛.com";
        PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
                new Intent(this, ActivityMain.class), 0);
        Notification notification = new Notification(R.drawable.default_icon,
                title, System.currentTimeMillis());
        notification.setLatestEventInfo(this, title, contents, contentIntent);
        // 100ms 延迟后，振动 250ms，停止 100ms 后振动 500ms
        notification.vibrate = new long[] { 100, 250, 100, 500 };
        notificationManager.notify(NOTIFICATIONS_ID, notification);
    }
}
```

上述代码是通过 Toast 实现的，它不需要用 NotificationManager 来管理，以上代码的处理流程如下。

(1) 实例化 Toast，每个 Toast 和一个 View 相关。

(2) 设置 Toast 的长短，通过“showToast(Toast.LENGTH_SHORT);”来设置 Toast 短时间显示，通过“showToast(Toast.LENGTH_LONG);”来设置 Toast 长时间显示。

(3) 通过函数 showToast 来显示 Toast，具体说明如下。

- 当单击“长时显示”按钮后，程序会长时间地显示提醒，并用 Notification 在状态栏中提示用户。
- 当单击“短时显示”按钮后，程序会短时间地显示提醒。

这样单击第二个 Button 后会执行上述处理程序，来到对应的新界面，新界面的布局文件是由 Activity_toast.xml 实现的。其主要代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:orientation="vertical" Android:layout_width="fill_parent"
    Android:layout_height="fill_parent">
    <Button Android:id="@+id/button1"
        Android:layout_width="wrap_content"
        Android:layout_height="wrap_content" Android:text="短时显示 Toast" />
```

```
    <Button Android:id="@+id/button2"
        Android:layout_width="wrap_content"
        Android:layout_height="wrap_content" Android:text="长时显示 Toast" />
</LinearLayout>
```

执行后新界面的效果如图 5-44 所示。

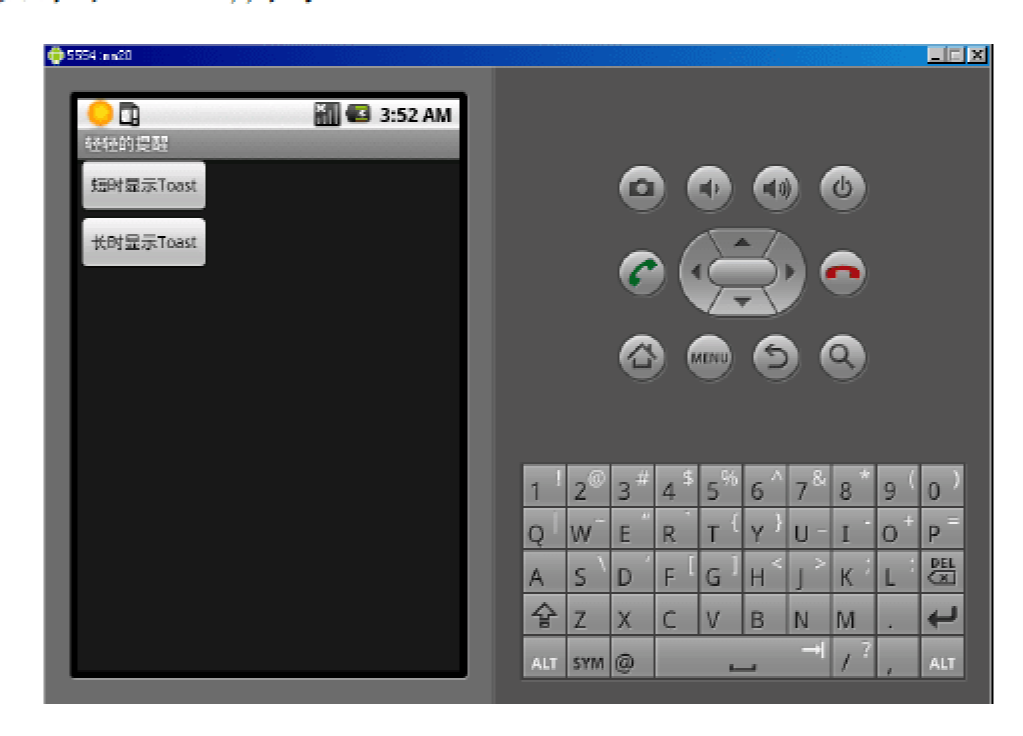


图 5-44　运行效果

当单击图 5-44 中的 Button 后，会根据上述处理文件实现某种效果。例如：当单击“短时显示 Toast”按钮后，会短时间显示一个提醒；当单击“长时显示 Toast”按钮后，会长时间显示一个提醒。两种方式的提醒界面都是相同的，具体效果如图 5-45 所示。

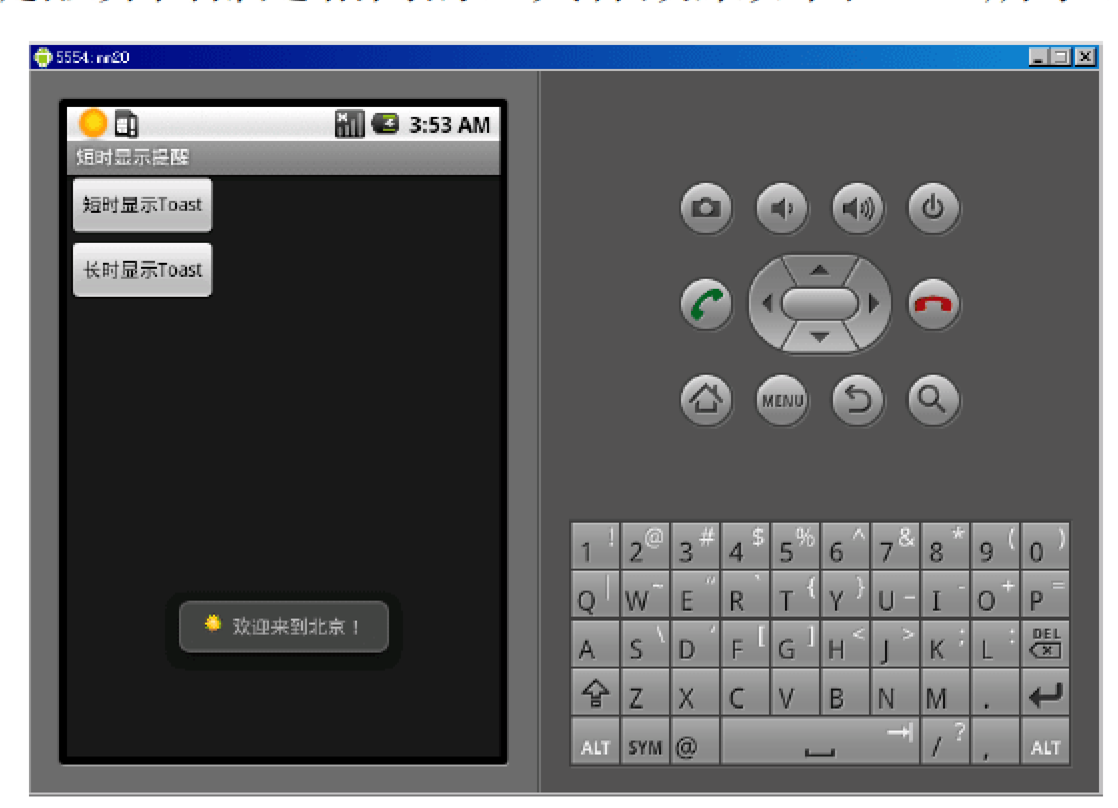


图 5-45　运行效果



第 6 章　相忘于江湖

通过上一章的学习，大家基本了解了 Android 组件的基本知识。本章将进一步讲解 Android 中 widget 的具体使用方法，并通过具体实例的实现过程讲解各个组件的具体使用流程，为读者步入本书后面知识的学习打下坚实的基础。

6.1　易 容 之 术

下午来到了少时经常听书的茶馆中，听了一段久违的评书。

要问天下谁第一，小李探花有飞刀！上回说到上官飞用易容术想欺骗李探花，但是没想到被探花一眼就识破了……

易容术，记得在秘籍中有易容术的记载，想到此处，我决定找个僻静的地方继续练功。

秘籍中写道：易者改变，容者容貌，所谓易容术便是改变人容貌的艺术。Android 的 widget 组件有易容术之妙，能够展现给我们各种不同的界面效果。秘籍中有 Android 易容术的详细心法，并且还提供了一个测试程序供我练习。

题　目	目　的	源码路径
演练 1	实战演练创建一个 widget 组件的过程	“光盘:\daima\6\Usewidget”文件夹

创建一个 Widget 组件的流程，具体如下。

第 1 步：打开 Eclipse，新建一个名为“ex_widgetdemo”的工程文件。

第 2 步：创建完毕后会自行创建一个 MainActivity，这是应用程序的入口，我们可以打开对应的文件 ex_widgetdemo.java。其主要代码如下所示：

```
package com.eoemobile.book.ex_widgetdemo;
import android.app.Activity;
import android.os.Bundle;
public class ex_widgetdemo extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);
    }
}
```

在上述代码中，主要功能是通过 onCreate 方法实现的，上述文件已经关联了一个模板文件 main.xml，这样，就可以继续添加需要的控件了，例如按钮、列表框、进度条和图片等。

6.1.1 最简单的按钮 Button

Button 是一个按钮控件，在日常应用时，当单击 Button 后会触发一个事件，这个事件会实现用户需要的功能。例如用户输入一些信息，单击“确定”或“取消”按钮后，会实现对应的功能。

注意：从本节开始，所有的演练都是基于 6.1 节所建工程的基础上进行的。

第 1 步：修改 main.xml 布局，添加一个 TextView 和一个 Button。

第 2 步：在 mainActivity.java 中用 findViewByID()获取 TextView 和 Button 资源。主要代码如下所示：

```
show= (TextView)findViewById(R.id.show TextView);
press=(Button)findViewById(R.id.Click_Button);
```

第 3 步：编写处理事件的主处理程序。具体代码如下所示：

```
press.setOnClickListener(new Button.OnClickListener(){
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                show.setText("Hi , Google Android!");
            }
        });
```

运行后将首先显示一个“按钮+文本”界面，当单击按钮后会执行单击事件，显示对应的文本提示，如图 6-1 所示。



图 6-1 执行效果

6.1.2 使用 TextView 控件显示文本

文本框控件 TextView 是使用最频繁的控件之一，其功能是在屏幕中输出一段文字。使用 TextView 需要通过如下 4 个步骤。

第 1 步：导入 TextView 包，然后在 mainActivity.java 中声明一个 TextView。具体代码如下所示：

```
import Android.widget.TextView;
private TextView mTextView01;
```

第 2 步：在 main.xml 中定义一个 TextView，然后设置 TextView 标签内容。具体代码如下所示：

```
<TextView Android:text="TextView01"
               Android:id="@+id/TextView01"
               Android:layout_width="wrap_content"
               Android:layout_height="wrap_content"
               Android:layout_x="61px"
               Android:layout_y="69px">
</TextView>
String str_2 = "欢迎来到Android 的TextView 世界...";
mTextView01.setText(str_2);
```

第 3 步：通过使用 findViewById()方法，来获取 main.xml 中的 TextView 值。具体代码如下所示：

```
mTextView01 = (TextView) findViewById(R.id.TextView01);
```

第 4 步：设置文本超级链接。具体代码如下所示：

```
<TextView
        Android:id="@+id/TextView02"
        Android:layout_width="wrap_content"
        Android:layout_height="wrap_content"
        Android:autoLink="all"
        Android:text="请访问Android 开发者：
        http://developer.Android.com/index.html">
</TextView>
```

使用 TextView 不但可以显示文本，而且可以设置文本的字体和颜色。

1. 使用颜色

可以使用 TextView 实现颜色变幻的效果，各个颜色的具体说明如下。

- Color.GRAY：灰色。
- Color.GREEN：绿色。
- Color.LTGRAY：浅灰色。
- Color.MAGENTA：红紫色。
- Color.RED：红色。
- Color.TRANSPARENT：透明。
- Color.WHITE：白色。
- Color.YELLOW：黄色。
- Color.BLACK：黑色。
- Color.BLUE：蓝色。
- Color.CYAN：青绿色。
- Color.DKGRAY：灰黑色。

下面新建一个工程，修改 mainActivity.java 文件，添加了 12 个 TextView 对象变量，一个 LinearLayout 对象变量、一个 WC 整数变量、一个 LinearLayout.LayoutParams 变量，具体实现代码如下所示：

```
package zyf.ManyColorME;
/*导入要使用的包*/
import Android.app.Activity;
import Android.graphics.Color;
import Android.os.Bundle;
import Android.widget.LinearLayout;
import Android.widget.TextView;
public class ManyColorME extends Activity {
    /** Called when the activity is first created. */
    /* 定义使用的对象 */
    private LinearLayout myLayout;
    private LinearLayout.LayoutParams layoutP;
    private int WC = LinearLayout.LayoutParams.WRAP_CONTENT;
    private TextView black_TV, blue_TV, cyan_TV, dkgray_TV,
                gray_TV, green_TV,ltgray_TV, magenta_TV, red_TV,
                transparent_TV, white_TV, yellow_TV;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /* 实例化一个 LinearLayout 布局对象 */
        myLayout = new LinearLayout(this);
        /* 设置 LinearLayout 的布局为垂直布局 */
        myLayout.setOrientation(LinearLayout.VERTICAL);
        /* 设置 LinearLayout 布局背景图片 */
        myLayout.setBackgroundResource(R.drawable.back);
        /* 加载主屏布局 */
        setContentView(myLayout);
        /* 实例化一个 LinearLayout 布局参数，用来添加 View */
        layoutP = new LinearLayout.LayoutParams(WC, WC);
        /* 构造实例化 TextView 对象 */
        constructTextView();
        /* 把 TextView 添加到 LinearLayout 布局中 */
        addTextView();
        /* 设置 TextView 文本颜色 */
        setTextViewColor();
        /* 设置 TextView 文本内容 */
        setTextViewText();
    }
    /* 设置 TextView 文本内容 */
    public void setTextViewText() {
        black_TV.setText("黑色");
        blue_TV.setText("蓝色");
        cyan_TV.setText("青绿色");
        dkgray_TV.setText("灰黑色");
        gray_TV.setText("灰色");
        green_TV.setText("绿色");
        ltgray_TV.setText("浅灰色");
        magenta_TV.setText("红紫色");
        red_TV.setText("红色");
        transparent_TV.setText("透明");
```

```
        white_TV.setText("白色");
        yellow_TV.setText("黄色");
    }
    /* 设置 TextView 文本颜色 */
    public void setTextViewColor() {
        black_TV.setTextColor(Color.BLACK);
        blue_TV.setTextColor(Color.BLUE);
        dkgray_TV.setTextColor(Color.DKGRAY);
        gray_TV.setTextColor(Color.GRAY);
        green_TV.setTextColor(Color.GREEN);
        ltgray TV.setTextColor(Color.LTGRAY);
        magenta_TV.setTextColor(Color.MAGENTA);
        red_TV.setTextColor(Color.RED);
        transparent_TV.setTextColor(Color.TRANSPARENT);
        white_TV.setTextColor(Color.WHITE);
        yellow_TV.setTextColor(Color.YELLOW);
    }
    /* 构造实例化 TextView 对象 */
    public void constructTextView() {
        black_TV = new TextView(this);
        blue_TV = new TextView(this);
        cyan_TV = new TextView(this);
        dkgray_TV = new TextView(this);
        gray_TV = new TextView(this);
        green_TV = new TextView(this);
        ltgray_TV = new TextView(this);
        magenta_TV = new TextView(this);
        red_TV = new TextView(this);
        transparent_TV = new TextView(this);
        white_TV = new TextView(this);
        yellow_TV = new TextView(this);
    }
    /* 把 TextView 添加到 LinearLayout 布局中 */
    public void addTextView() {
        myLayout.addView(black TV, layoutP);
        myLayout.addView(blue_TV, layoutP);
        myLayout.addView(cyan_TV, layoutP);
        myLayout.addView(dkgray_TV, layoutP);
        myLayout.addView(gray_TV, layoutP);
        myLayout.addView(green_TV, layoutP);
        myLayout.addView(ltgray_TV, layoutP);
        myLayout.addView(magenta_TV, layoutP);
        myLayout.addView(red TV, layoutP);
        myLayout.addView(transparent_TV, layoutP);
        myLayout.addView(white_TV, layoutP);
        myLayout.addView(yellow_TV, layoutP);
    }
}
```

上述代码执行后的效果如图 6-2 所示。

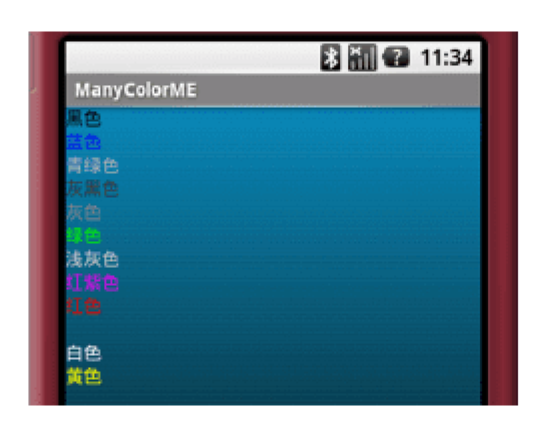


图 6-2　执行后的效果

2. 实现静态域字体

在计算机领域，字体风格用 Typeface 种类表示，有如下两种类型。

1)　int Style 类型

int Style 类型的具体说明如表 6-1 所示。

表 6-1　int Style 类型说明

字　体	说　明
BOLD	粗体
BOLD_ITALIC	粗斜体
ITALIC	斜体
NORMAL	普通字体

2)　Typeface 类型

Typeface 类型的具体说明如表 6-2 所示。

表 6-2　Typeface 类型说明

字　体	说　明
DEFAULT	默认字体
DEFAULT_BOLD	默认粗体
MONOSPACE	单间隔字体
SANS_SERIF	无衬线字体
SERIF	衬线字体

我决定实际演练一下，先修改 mianActivity.java 文件以实现多种字体显示，主要实现代码如下所示：

```
package zyf.TypefaceStudy;
/*导入要使用的包*/
……………………………………………
public class TypefaceStudy extends Activity {
    /** Called when the activity is first created. */
    /*
     * Android.graphics.Typeface java.lang.Object
       Typeface 类指定一个字体和固有风格。
     * 该类用于绘制，与可选绘制设置一起使用，
    如 textSize、textSkewX、textScaleX 当绘制(测量)时来指定如何显示文本。
     */
```

```
/* 定义实例化一个布局大小，用来添加 TextView */
final int WRAP_CONTENT = ViewGroup.LayoutParams.WRAP_CONTENT;
/* 定义 TextView 对象 */
private TextView bold_TV, bold_italic_TV, default_TV,
                default_bold_TV,italic_TV,monospace_TV,
                normal_TV,sans_serif_TV,serif_TV;
/* 定义 LinearLayout 布局对象 */
private LinearLayout linearLayout;
/* 定义 LinearLayout 布局参数对象 */
private LinearLayout.LayoutParams linearLayoutParams;
@Override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    /* 定义实例化一个 LinearLayout 对象 */
    linearLayout = new LinearLayout(this);
    /* 设置 LinearLayout 布局为垂直布局 */
    linearLayout.setOrientation(LinearLayout.VERTICAL);
    /*设置布局背景图*/
    linearLayout.setBackgroundResource(R.drawable.back);
    /* 加载 LinearLayout 为主屏布局，显示 */
    setContentView(linearLayout);
    /* 定义实例化一个 LinearLayout 布局参数 */
    linearLayoutParams = new LinearLayout.LayoutParams(WRAP_CONTENT,WRAP_
    CONTENT);
    constructTextView();
    setTextSizeOf();
    setTextViewText() ;
    setStyleOfFont();
    setFontColor();
    toAddTextViewToLayout();
}
public void constructTextView() {
    /* 实例化 TextView 对象 */
    bold_TV = new TextView(this);
    bold_italic_TV = new TextView(this);
    default_TV = new TextView(this);
    default_bold_TV = new TextView(this);
    italic_TV = new TextView(this);
    monospace_TV=new TextView(this);
    normal_TV=new TextView(this);
    sans_serif_TV=new TextView(this);
    serif_TV=new TextView(this);
}
public void setTextSizeOf() {
    // 设置绘制的文本大小，该值必须大于 0
    bold_TV.setTextSize(24.0f);
    bold_italic_TV.setTextSize(24.0f);
    default_TV.setTextSize(24.0f);
    default_bold_TV.setTextSize(24.0f);
    italic_TV.setTextSize(24.0f);
    monospace_TV.setTextSize(24.0f);
    normal_TV.setTextSize(24.0f);
```

```
        sans_serif_TV.setTextSize(24.0f);
        serif_TV.setTextSize(24.0f);
    }
    public void setTextViewText() {
        /* 设置文本 */
        bold_TV.setText("BOLD");
        bold_italic_TV.setText("BOLD_ITALIC");
        default_TV.setText("DEFAULT");
        default_bold_TV.setText("DEFAULT_BOLD");
        italic_TV.setText("ITALIC");
        monospace_TV.setText("MONOSPACE");
        normal_TV.setText("NORMAL");
        sans_serif_TV.setText("SANS_SERIF");
        serif_TV.setText("SERIF");
    }
    public void setStyleOfFont() {
        /* 设置字体风格 */
        bold_TV.setTypeface(null, Typeface.BOLD);
        bold_italic_TV.setTypeface(null, Typeface.BOLD_ITALIC);
        default_TV.setTypeface(Typeface.DEFAULT);
        default_bold_TV.setTypeface(Typeface.DEFAULT_BOLD);
        italic_TV.setTypeface(null, Typeface.ITALIC);
        monospace_TV.setTypeface(Typeface.MONOSPACE);
        normal_TV.setTypeface(null, Typeface.NORMAL);
        sans_serif_TV.setTypeface(Typeface.SANS_SERIF);
        serif_TV.setTypeface(Typeface.SERIF);
    }
    public void setFontColor() {
        /* 设置文本颜色 */
        bold_TV.setTextColor(Color.BLACK);
        bold_italic_TV.setTextColor(Color.CYAN);
        default_TV.setTextColor(Color.GREEN);
        default_bold_TV.setTextColor(Color.MAGENTA);
        italic_TV.setTextColor(Color.RED);
        monospace_TV.setTextColor(Color.WHITE);
        normal_TV.setTextColor(Color.YELLOW);
        sans_serif_TV.setTextColor(Color.GRAY);
        serif_TV.setTextColor(Color.LTGRAY);
    }
    public void toAddTextViewToLayout() {
        /* 把 TextView 加入 LinearLayout 布局中 */
        linearLayout.addView(bold_TV, linearLayouttParams);
        linearLayout.addView(bold_italic_TV, linearLayouttParams);
        linearLayout.addView(default_TV, linearLayouttParams);
        linearLayout.addView(default_bold_TV, linearLayouttParams);
        linearLayout.addView(italic_TV, linearLayouttParams);
        linearLayout.addView(monospace_TV, linearLayouttParams);
        linearLayout.addView(normal_TV, linearLayouttParams);
        linearLayout.addView(sans_serif_TV, linearLayouttParams);
        linearLayout.addView(serif_TV, linearLayouttParams);
    }
}
```

上述代码运行后的效果如图 6-3 所示。



图 6-3　运行效果

3. 在代码中更改 TextView 文字颜色

秘籍上说也可以在代码中更改 TextView 文字的颜色，具体实现流程如下。

第 1 步：在文件 main.xml 中编写布局代码。

第 2 步：根据 ID 获取 TextView 和 Resources 对象。具体代码如下所示：

```
TextView text_A=(TextView)findViewById(R.id.TextView01);
TextView text_B=(TextView)findViewById(R.id.TextView02);
Resources myColor_R=getBaseContext().getResources();
```

第 3 步：添加文件 drawable.xml，在此添加一个 white 颜色值，具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="white">#ffffffff</color>
</resources>
```

第 4 步：获取 Drawable 对象，然后设置文本背景颜色。具体代码如下所示：

```
Drawable myColor_D=myColor_R.getDrawable(R.color.white);
text_A.setBackgroundDrawable(myColor_D);
```

第 5 步：利用 Android.graphics.Color 的颜色静态变量来改变文本颜色。具体代码如下所示：

```
text_A.setTextColor(Android.graphics.Color.GREEN);
```

第 6 步：利用 Color 的静态常量来设置文本颜色。具体代码如下所示：

```
text_B.setTextColor(Color.RED);
```

6.1.3　收回我说的话

现实中的你，如果说错了一句话可以收回。同样，安卓屏幕中的文本也可以修改，我们可

以使用编辑框控件 EditView 生成一个可编辑的文本框，秘籍中提供了如下的操作步骤。

第 1 步：在程序的主窗口界面中添加一个 EditView 按钮，然后设定其监听器在接收到单击事件时程序打开 EditView 的界面。

第 2 步：编写事件处理文件 EditTextActivity.java。主要代码如下所示：

```
public class EditTextActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("EditTextActivity");
        setContentView(R.layout.editview);
        find_and_modify_text_view();
    }
    private void find and modify text view() {
        Button get edit view button = (Button) findViewById(R.id.get edit
        view_button);

    get_edit_view_button.setOnClickListener(get_edit_view_button_listener);
    }
    private Button.OnClickListener get_edit_view_button_listener = new Button.
    OnClickListener() {
      /**响应代码，显示 EditText 中的值**/
        public void onClick(View v) {
            EditText edit_text = (EditText) findViewById(R.id.edit_text);
            CharSequence edit_text_value = edit_text.getText();
            setTitle("EditText 的值:"+edit_text_value);
        }
    };
}
```

上述代码执行后，将首先显示默认的文本和输入框，初始效果如图 6-4 所示。输入一段文本，单击“获取 EditView 的值”按钮后，会获取输入的文字并显示输入的文字，运行后的效果如图 6-5 所示。



图 6-4　初始效果



图 6-5　运行效果

6.1.4　你可以有多个选择

虽然成功的路有很多条，但是我们需要自己选择要走的路。在手机屏幕中也有多个选择，CheckBox 控件能够为用户提供输入的信息，用户可以一次性选择多个选项。在 Android 中使用 CheckBox 控件时，也需要在 XML 文件中定义，秘籍中提供了如下操作步骤。

第 1 步：设计 XML 文件 check_box.xml，插入 4 个选项供用户选择。

第 2 步：编写事件处理文件 CheckBoxActivity.java 的代码。主要代码如下所示：

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("CheckBoxActivity");
        setContentView(R.layout.check_box);
        find and modify text view();
    }
    private void find_and_modify_text_view() {
        plain_cb = (CheckBox) findViewById(R.id.plain_cb);
        serif_cb = (CheckBox) findViewById(R.id.serif_cb);
        italic_cb = (CheckBox) findViewById(R.id.italic_cb);
        bold_cb = (CheckBox) findViewById(R.id.bold_cb);
        Button get_view_button = (Button) findViewById(R.id.get_view_button);
        get_view_button.setOnClickListener(get_view_button_listener);
    }

    private Button.OnClickListener get_view_button_listener = new Button.
    OnClickListener() {
        public void onClick(View v) {
            String r = "";
            if (plain_cb.isChecked()) {
                r = r + "," + plain_cb.getText();
            }
            if (serif_cb.isChecked()) {
                r = r + "," + serif_cb.getText();
            }
            if (italic_cb.isChecked()) {
                r = r + "," + italic_cb.getText();
            }
            if (bold_cb.isChecked()) {
                r = r + "," + bold cb.getText();
            }
            setTitle("Checked: " + r);
        }
    };
}
```

在上述代码中，把用户选中的选项值显示在 Title 上面。执行后，将首先显示 4 个选项值供用户选择，初始效果如图 6-6 所示；用户选择某些选项并单击“获取 CheckBox 的值”按钮后，文本提示用户选择的选项，运行效果如图 6-7 所示。



图 6-6　初始效果

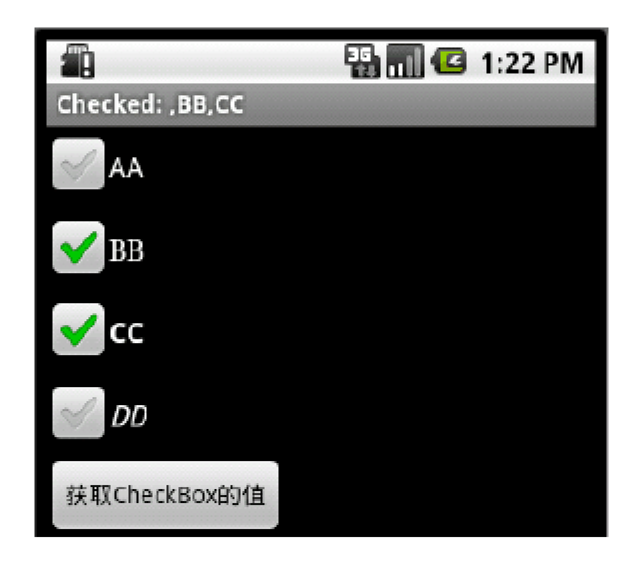


图 6-7　运行效果

6.1.5　你只能选择一个

现实太残酷，有时候你只有一条路可以走。同样，在安卓应用中，有时只能有一个选项供我们选择。单选按钮控件 RadioGroup 是和复选框控件 CheckBox 相对应的，但是它只能供用户选择一个选项。在 Android 中，使用 RadioGroup 控件也需要在 XML 文件中定义，秘籍中提供了如下操作步骤。

第 1 步：设计 XML 文件 radio_group.xml，插入 4 个选项供用户选择。

第 2 步：编写事件处理文件 RadioGroupActivity.java 的代码。主要代码如下所示：

```
@Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.radio_group);
       setTitle("RadioGroupActivity");
       mRadioGroup = (RadioGroup) findViewById(R.id.menu);
       Button clearButton = (Button) findViewById(R.id.clear);
       clearButton.setOnClickListener(this);
   }
```

当用户单击“清除”按钮后将使用 setTitle 修改 Title 为“RadioGroupActivity”，然后会获取 RadioGroup 对象和按钮对象。

至此，整个实例设计完毕。执行后，将首先显示 4 个选项值供用户选择，初始效果如图 6-8 所示；用户选择一个选项并单击“清除”按钮后，将会清除选择的选项，运行效果如图 6-9 所示。

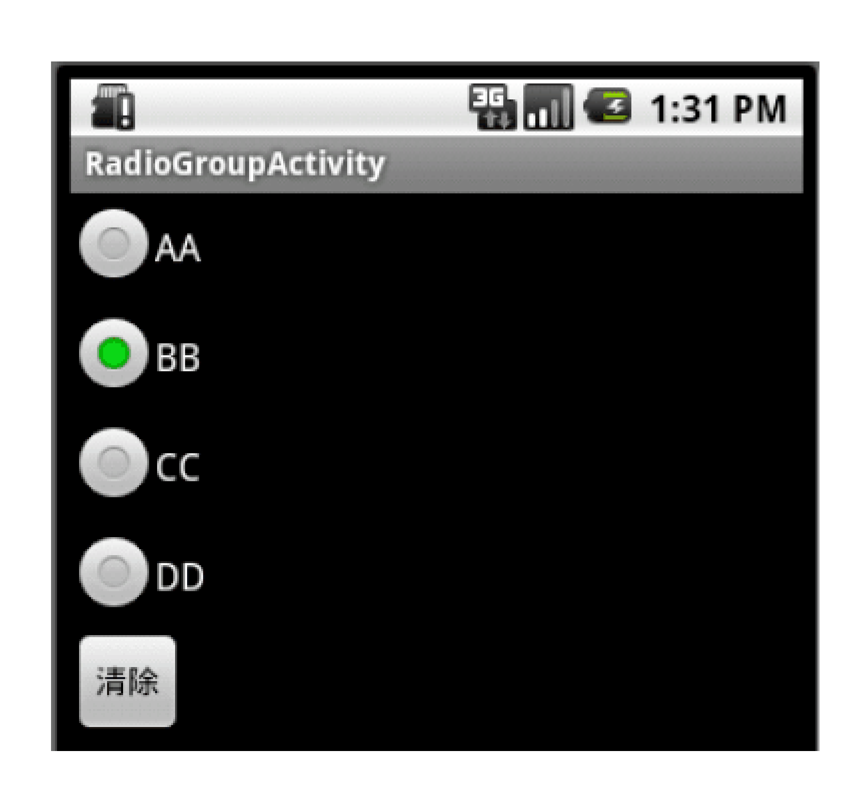


图 6-8　初始效果

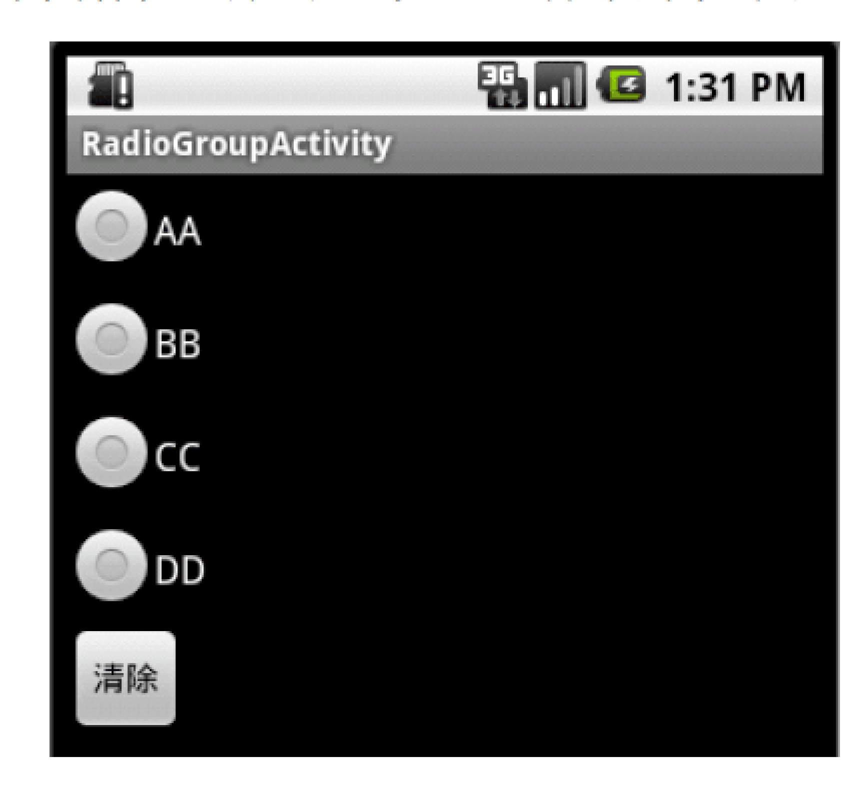


图 6-9　运行效果

6.1.6　下拉列表控件 Spinner

下拉列表控件 Spinner 能够提供下拉选择样式的输入框，用户不需要输入数据，只要选择一个选项后即可在输入框中完成数据输入。秘籍中使用下拉列表控件 Spinner 的操作步骤如下。

第 1 步：在 Android 中使用 Spinner 控件之前，首先需要在 main.xml 中添加一个按钮，单击这个按钮后会启动这个 SpinnerActivity 文件。

第 2 步：在文件 MainActivity.java 中编写上述按钮的处理事件代码。具体代码如下所示：

```
private Button.OnClickListener spinner_button_listener = new Button. OnClickListener() {
```

```
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, SpinnerActivity.class);
            startActivity(intent);
        }
    };
```

在上述代码中，启动了 SpinnerActivity 文件，该文件可以展示 Spinner 组件的界面。在具体实现上，先创建了 SpinnerActivity 的 Activity，然后修改了其 onCreate 方法，指定其对应模板为 spinner.xml。文件 SpinnerActivity.java 中对应的代码如下所示：

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("SpinnerActivity");
        setContentView(R.layout.spinner);
        find_and_modify_view();
    }
```

第 3 步：编写文件 spinner.xml，添加了 2 个 TextView 控件和 2 个 Spinner 控件。

第 4 步：在文件 AndroidManifest.xml 中引入了另外一个 Activity：SpinnerActivity。

```
<activity Android:name="SpinnerActivity"></activity>
```

经过上述处理后，即可在界面中生成一个简单的单选选项界面，但是在列表中并没有选项值。如果要在下拉列表中实现可供用户选择的选项值，还需要填充一些数据。

第 5 步：载入列表数据，首先定义需要载入的数据，然后在 onCreate 方法中通过调用 find_and_modify_view()来完成数据的载入。文件 SpinnerActivity.java 中实现上述功能的具体代码如下所示：

```
private static final String[] mCountries = { "China" ,"Russia", "Germany",
    "Ukraine", "Belarus", "USA" };
private void find_and_modify_view() {
        spinner_c = (Spinner) findViewById(R.id.spinner_1);
        allcountries = new ArrayList<String>();
        for (int i = 0; i < mCountries.length; i++) {
        allcountries.add(mCountries[i]);
        }
        aspnCountries = new ArrayAdapter<String>(this,
        Android.R.layout.simple_spinner_item, allcountries);
        aspnCountries.setDropDownViewResource(Android.R.layout.simple
        _spinner_dropdown_item);
        spinner_c.setAdapter(aspnCountries);
```

在上述代码中，将定义的 mCountries 数据载入到了 Spinner 组件中。

第 6 步：在文件 spinner.xml 中预定义数据，即在 spinner.xml 模板中再添加一个 Spinner 组件，然后在文件 SpinnerActivity.java 中初始化它的值。具体代码如下所示：

```
spinner_2 = (Spinner) findViewById(R.id.spinner_2);
  ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
  this, R.array.countries, Android.R.layout.simple_spinner_item);
  adapter.setDropDownViewResource(Android.R.layout.simple spinner dropdown
  item);
```

```
    spinner_2.setAdapter(adapter);
```

在上述代码中，将 R.array.countries 对应值载入到了 spinner_2 中去，而 R.array.countries 的对应值是在文件 array.xml 中预先定义的，文件 array.xml 的具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <!-- Used in Spinner/spinner_2.java -->
   <string-array name="countries">
      <item>AA</item>
      <item>BB</item>
      <item>CC</item>
      <item>DD</item>
      <item>EE</item>
      <item>FF</item>
   </string-array>
</resources>
```

在上述代码中，预定义了一个名为“countries”的数组。至此，整个演练就完美结束了。执行后，将首先显示 2 个下拉列表表单，初始效果如图 6-10 所示；用户单击一个下拉列表表单后面的▼按钮时，会弹出一个由 Spinner 组件实现的下拉选项框，运行效果如图 6-11 所示；当选择一个选项后，选项值会自动出现在输入表单中，效果如图 6-12 所示。

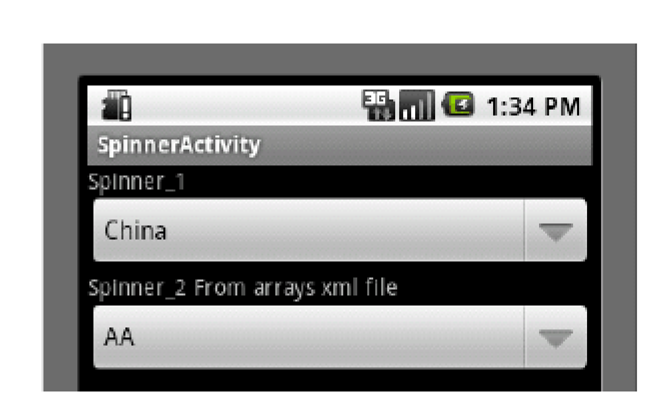


图 6-10　初始效果

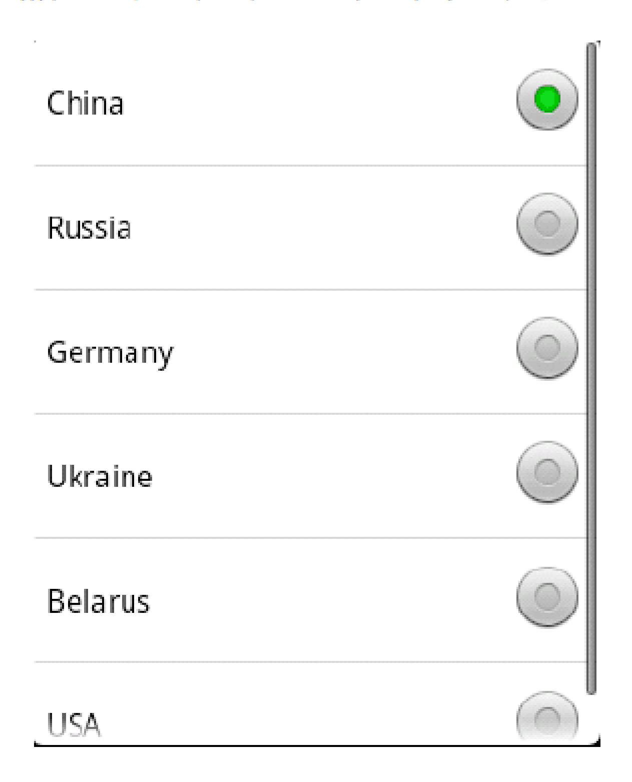


图 6-11　运行效果

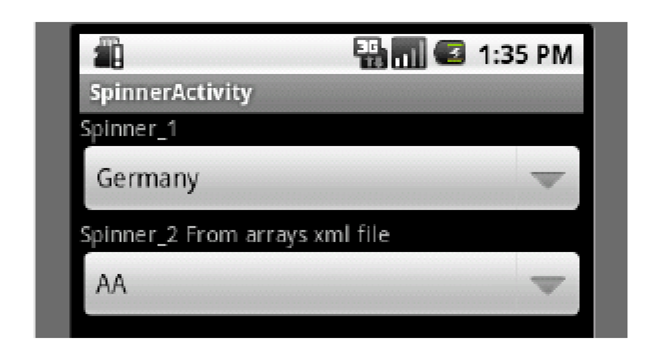


图 6-12　选择值自动出现在表单中

6.1.7　体验全自动带来的魅力

我十分憧憬将来出行时不用自己走，吃饭只需要张嘴……当然这只是一个美好的希望而已。在安卓中有一个和自动化有关的控件——AutoCompleteTextView，其功能是帮助用户自动输入数据，例如当用户输入一个字符后，能够根据这个字符的提示显示出与之相关的数据。此应用在搜索派的引擎产品中比较常见，例如用户在百度中输入关键字“杜拉拉”后，会在下拉列表中自动显示出相关的关键词，效果如图 6-13 所示。



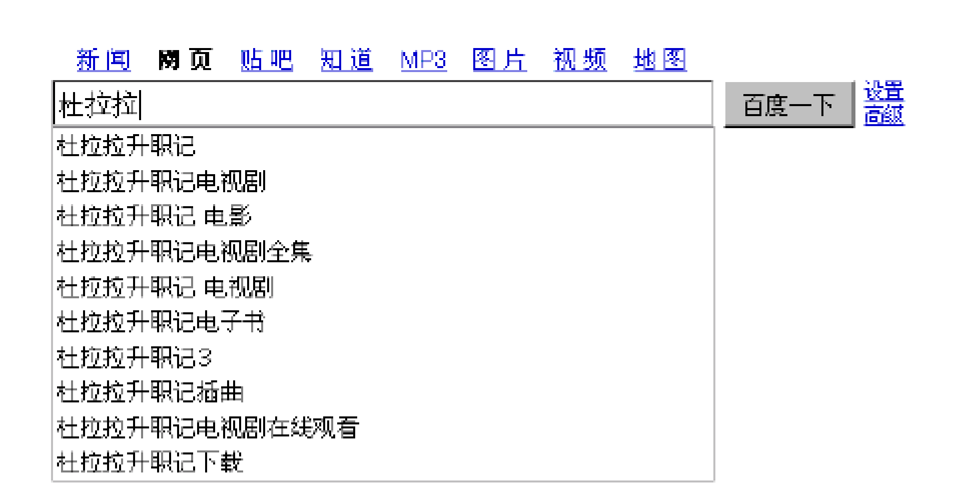


图 6-13　百度的输入提示框

秘籍中说：安卓手机应用中，AutoCompleteTextView 可以实现和图 6-13 类似的功能。

使用 AutoCompleteTextView 控件的基本流程如下所示。

第 1 步：在 main.xml 布局中添加一个 TextView、一个 AutoCompleteTextView、一个 Button 按钮。

第 2 步：修改 mainActivity.java 文件，添加自动完成功能处理事件。具体代码如下所示：

```
public class Ex_Ctrl_13ME extends Activity {
    /** Called when the activity is first created. */
    /*定义要使用的类对象*/
    private String[] normalString =
           new String[] {
            "Android", "Android Blog","Android Market", "Android SDK",
          "Android AVD","BlackBerry","BlackBerry JDE", "Symbian",
          "Symbian Carbide", "Java 2ME","Java FX", "Java 2EE",
          "Java 2SE", "Mobile", "Motorola", "Nokia", "Sun",
            "Nokia Symbian", "Nokia forum", "WindowsMobile", "Broncho",
            "Windows XP", "Google", "Google Android ", "Google 浏览器",
            "IBM", "MicroSoft", "Java", "C++", "C", "C#", "J#", "VB" };
    @SuppressWarnings("unused")
    private TextView show;
    private AutoCompleteTextView autoTextView;
    private Button clean;
    private ArrayAdapter<String> arrayAdapter;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /*装入主屏布局 main.xml*/
        setContentView(R.layout.main);
        /*从 XML 中获取 UI 元素对象*/
        show = (TextView) findViewById(R.id.TextView_InputShow);
        autoTextView =
        (AutoCompleteTextView) findViewById(R.id.AutoCompleteTextView_input);
        clean = (Button) findViewById(R.id.Button_clean);
        /*实现一个适配器对象，用来给自动完成输入框添加自动装入的内容*/
        arrayAdapter = new ArrayAdapter<String>(this,
                Android.R.layout.simple_dropdown_item_1line, normalString);
        /*给自动完成输入框添加内容适配器*/
```

```
        autoTextView.setAdapter(arrayAdapter);
        /*给清空按钮添加单击事件处理监听器*/
        clean.setOnClickListener(new Button.OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                /*清空*/
                autoTextView.setText("");
            }
        });
    }
}
```

经过上述的简单操作，便成功地使用了 AutoCompleteTextView 控件。执行后，当在表单中输入数据后，会根据预先准备的数据进行提示，效果如图 6-14 所示。





图 6-14　输入数据

6.1.8　那些年，那些事

时间是金子，这是听师傅说的，所以我做任何事时都很注重效率。秘籍中说道：在安卓中能够实现和日期相关操作的控件是 DatePicker，这是一个日期选择器控件，其功能是为用户提供快速选择日期的方法。我知道日期的格式是“年—月—日”，在很多系统中都为用户提供了日期选择表单，这样不用用户输入具体的日期，只需用鼠标单击即可完成日期的设置。

使用日期选择器控件 DatePicker 的具体操作流程如下。

第 1 步：在 main.xml 中添加一个按钮，用于打开 DatePicker 界面。

第 2 步：定义上述按钮响应处理事件。具体代码如下所示：

```
private Button.OnClickListener date_picker_button_listener = new Button.
OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, DatePickerActivity.class);
            startActivity(intent);
        }
    };
```

这样，当单击 DatePicker 按钮后，会跳转到 DatePickerActivity 上。当创建一个 Activity 组件后，需要在其 onCreate 方法中指定需要绑定的模板文件为 date_picker.xml。

第 3 步：在文件 DatePickerActivity.java 中编写 onCreate 的实现代码，具体代码如下所示：

```
public class DatePickerActivity extends Activity {
    /** Called when the activity is first created. */
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("CheckBoxActivity");
        setContentView(R.layout.date_picker);
        DatePicker dp =  (DatePicker)this.findViewById(R.id.date_picker);
        dp.init(2010, 5, 17, null);
        }
    }
```

在上述代码中，设置了开始时间为 2010 年 5 月 17 日。

第 4 步：在文件 date_picker.xml 中添加 DatePicker 组件。主要代码如下所示：

```
    <DatePicker
        Android:id="@+id/date_picker"
        Android:layout_width="wrap_content"
        Android:layout_height="wrap_content"
    />
```

在上述代码中，设置了 DatePicker 控件的 ID 为“date_picker”，其宽度和高度都为自适应。

第 5 步：在文件 AndroidManifest.xml 中添加 Activity 声明。具体代码如下所示：

```
<activity Android:name="DatePickerActivity" />
```

至此，整个实例就设计完成。执行后，将首先显示设置的起始日期，初始效果如图 6-15 所示；分别单击月、日、年上面的“+”或下面的“-”后，将会自动显示更改后的月、日、年，改变后的效果如图 6-16 所示。

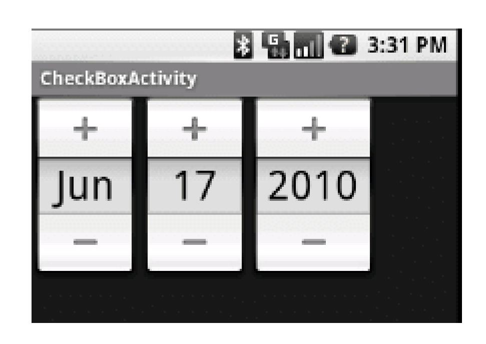


图 6-15　初始效果

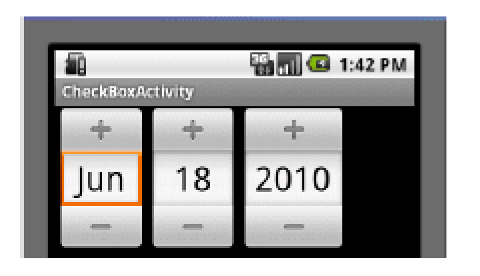


图 6-16　改变后的效果

6.1.9　生命的意义

秘籍中说道：除了 DatePicker 之外，TimePicker 控件也能实现和时间相关的功能。同 DatePicker 控件的功能类似，其功能是为用户提供快速选择时间的方法。

使用时间选择器 TimePicker 控件的基本操作流程如下。

第 1 步：在 main.xml 文件中添加一个 button 按钮。

第 2 步：编写响应上述按钮“time_picker”的事件代码。具体代码如下所示：

```
private Button.OnClickListener time_picker_button_listener = new Button.
OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, TimePickerTimePicker.class);
            startActivity(intent);
        }
};
```

通过上述代码，当单击按钮“time_picker”后会跳转到 TimePickerActivity 上。

第 3 步：创建一个 Activity，然后在其 onCreate 方法中指定需要绑定的模板为 time_picker.xml。文件 TimePickerActivity.java 中 onCreate 方法的实现代码如下所示：

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("TimePickerActivity");
        setContentView(R.layout.time_picker);
        TimePicker tp =  (TimePicker)this.findViewById(R.id.time_picker);
        tp.setIs24HourView(true);
    }
```

在上述代码中，首先指定了对应的布局模板是 time_picker.xml，然后获取了其中的 TimePicker 控件。

第 4 步：在文件 time_picker.xml 中定义 TimePicker。具体的代码如下所示：

```
<TimePicker
 Android:id="@+id/time_picker"
 Android:layout_width="wrap_content"
 Android:layout_height="wrap_content"/>
```

至此，整个演练就完美结束了。执行后，将首先显示设置的起始时间，分别单击时间上面的“+”或下面的“-”后，将会自动显示更改后的时间，运行效果如图 6-17 所示。

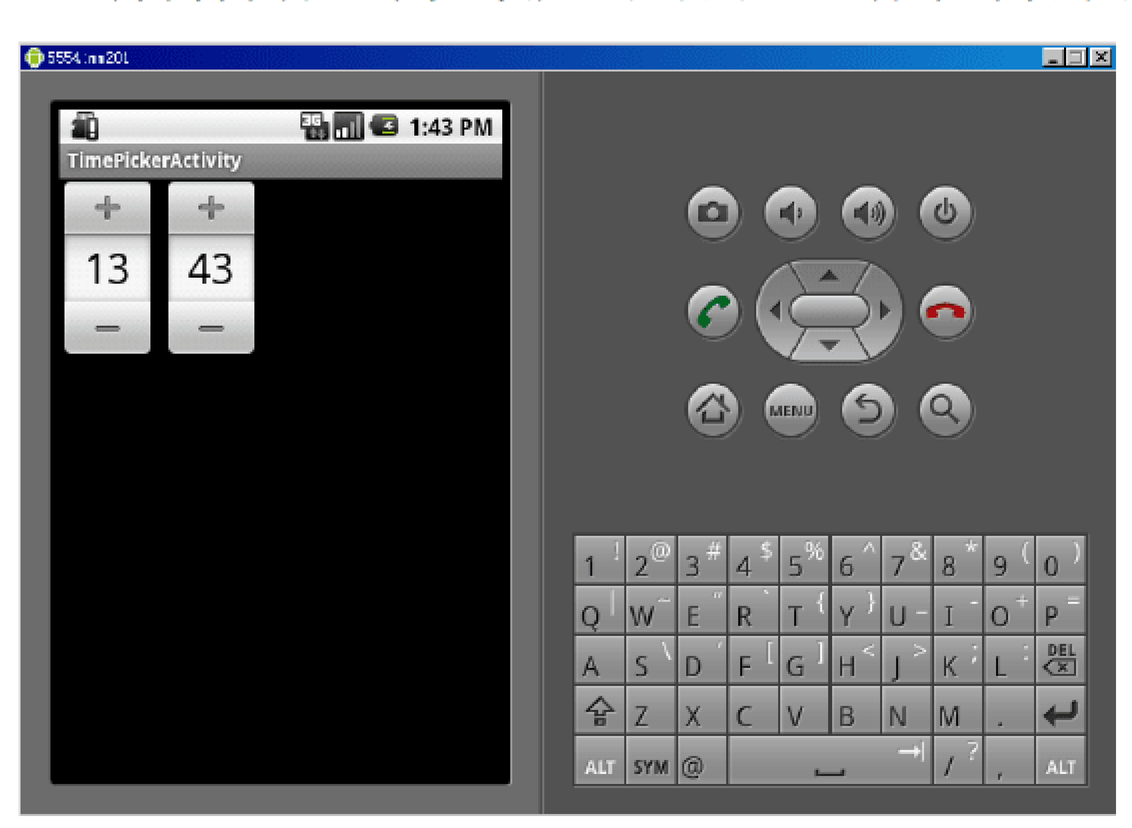


图 6-17　运行效果

6.1.10　滚滚黄河水

我终于见到了魂牵梦绕的黄河，黄河之水曲曲折折奔流到大海。看着黄河水的曲折，忽然想起了一个一直困扰我的问题：手机屏幕有限，能否也有一个类似网页滚动条的东西，这样就能够在有限的屏幕中显示无限的内容了。翻开秘籍，我找到了实现这个功能的答案。

滚动视图控件 ScrollView 的功能是：在手机屏幕中生成一个滚动样式的显示方式，这样即使内容超出了屏幕的大小，也能通过滚动的方式供用户浏览。使用滚动视图控件 ScrollView 的方法比较简单，只需在 LinearLayout 外面增加一个 ScrollView 控件即可，具体代码如下所示：

```
<ScrollView xmlns:Android="http://schemas.Android.com/apk/res/Android"
      Android:layout_width="fill_parent"
```

```
    Android:layout height="wrap content"
>
```

在上述代码中，将滚动视图控件 ScrollView 放在了 LinearLayout 的外面，这样当 LinearLayout 中的内容超过屏幕大小时，会实现滚动浏览功能。程序运行后的效果如图 6-18 所示。

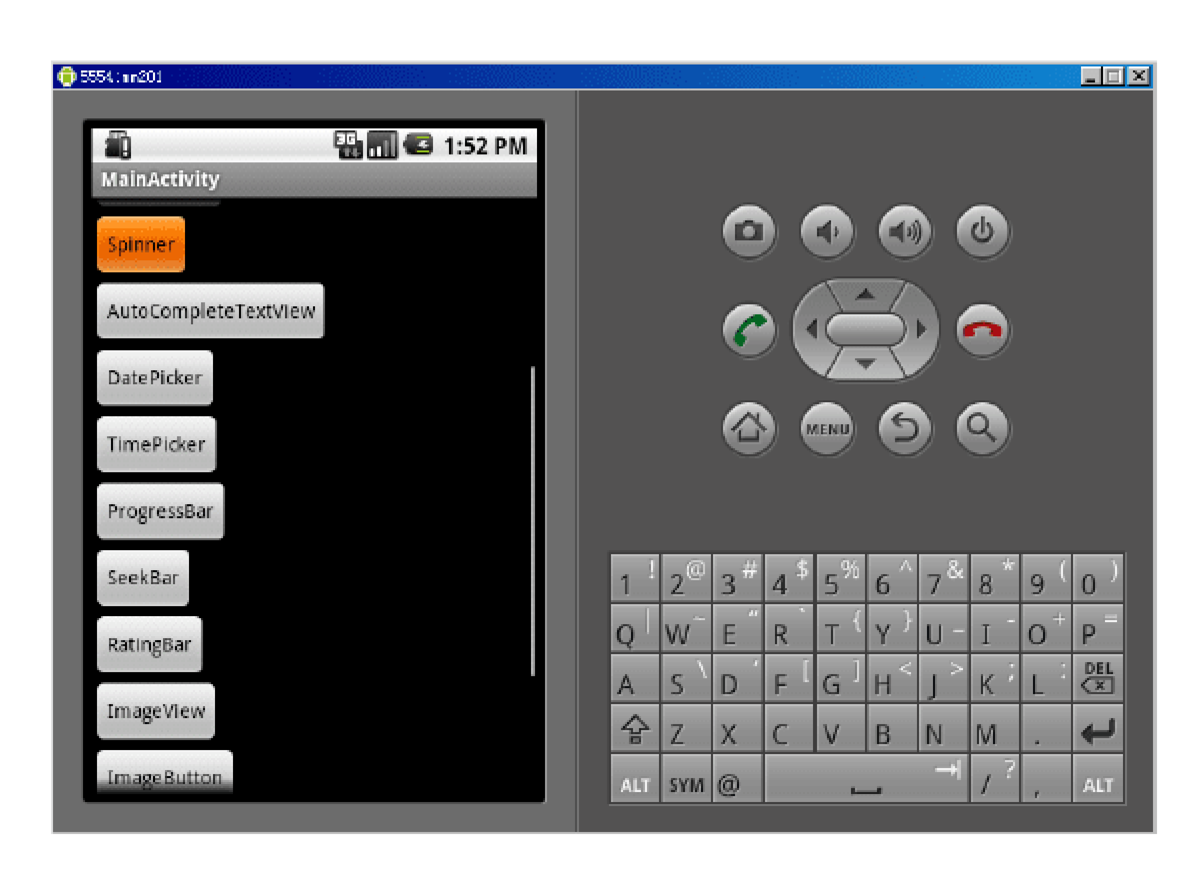


图 6-18　运行效果

6.1.11　不再让你焦虑

曾经有一段时间我很焦虑，做任何事都没有耐心，师傅发现了我的问题，让我静心修炼养性之道。其实在安卓中，有时候打开一个界面会消耗很多时间，为此推出了一个控件来消除用户的焦虑。在秘籍中是这样介绍的：进度条控件 ProgressBar 的功能是以图像化的方式显示某个过程的进度，这样做的好处是能够更加直观地显示进度。进度条在计算机领域中很常见，例如软件的安装过程一般都会使用进度条。

使用进度条控件 ProgressBar 的基本操作流程如下。

第 1 步：在 main.xml 中增加一个激活 ProgressBar 控件的按钮。

第 2 步：编写单击按钮的事件处理程序，用于打开进度条界面。文件 MainActivity.java 中的对应代码如下所示：

```
private  Button.OnClickListener  progress_bar_button_listener  =  new  Button.
OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, ProgressBarActivity.class);
            startActivity(intent);
        }
};
```

通过上述代码，当单击在 main.xml 中增加的按钮后会启动 ProgressBarActivity。

第 3 步：编写文件 ProgressBarActivity.java，用于设置其对应的布局文件为 Progress_Bar.xml。具体代码如下所示：

```
public class ProgressBarActivity extends Activity {
    CheckBox plain_cb;
    CheckBox serif_cb;
```

```
    CheckBox italic cb;
    CheckBox bold_cb;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("ProgressBarActivity");
        setContentView(R.layout.progress_bar);
    }
}
```

第 4 步：编写文件 Progress_Bar.xml，插入 2 个 ProgressBar 控件，其中设置第一个为环形进度条、第二个为水平进度样式。然后设置第一个进度到 50，第二个进度到 75。

至此，整个演练就全部结束了。执行后将显示对应样式的进度条，运行效果如图 6-19 所示。

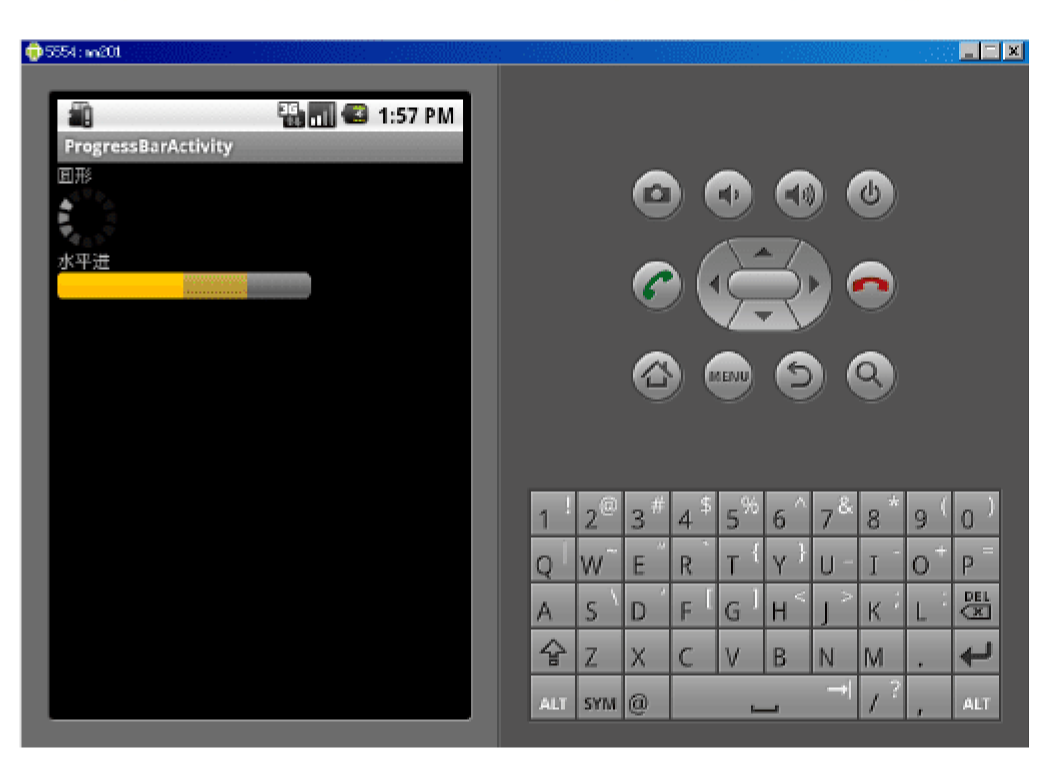


图 6-19　运行效果

6.1.12　拖动你的命运

江湖上有太多不可思议的事情，几乎天天上演一夜间灰飞烟灭或一夜间神秘崛起的故事。要想成功，有时也需要不走寻常路，我称之为拖动命运。在安卓中有一个能拖动发展的进度条，秘籍中的描述为：拖动条控件 SeekBar 的功能是通过拖动某个进程来直观地显示进度。现实中最常见的拖动条应用是播放器的播放进度，我们可以通过拖动来设置进度。

使用拖动条控件 SeekBar 的基本操作流程如下。

第 1 步：在文件 main.xml 中插入一个按钮，并为按钮处理事件编写代码。具体代码如下所示：

```
private Button.OnClickListener seek_bar_button_listener = new Button.
OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, SeekBarActivity.class);
            startActivity(intent);
        }
};
```

通过上述代码，当用户单击按钮后会跳转到 SeekBarActivity。

第 2 步：创建一个 Activity，为其指定模板 seek_bar.xml。文件 seek_bar.xml 的具体代码如

下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:orientation="vertical" Android:layout_width="fill_parent"
    Android:layout_height="wrap_content">
<TextView
       Android:layout_width="wrap_content"
       Android:layout_height="wrap_content"
       Android:text="SeekBar" />
       <SeekBar
       Android:id="@+id/seek"
       Android:layout_width="fill_parent"
       Android:layout_height="wrap_content"
       Android:max="100"
       Android:thumb="@drawable/seeker"
       Android:progress="50"/>
</LinearLayout>
```

在上述代码中，定义了一个 SeekBar 控件，设置了其 ID 为 seek，并设定了宽度为布满屏幕显示，显示的最大值为 100。

第 3 步：在文件 AndroidManifest.xml 中增加对 SeekBarActivity 的声明。对应的代码如下所示：

```
<activity Android:name="SeekBarActivity" />
```

至此，整个实例设计完毕。执行后将显示对应样式的进度条，用户可以通过鼠标来拖动进度条的位置，运行结果如图 6-20 所示。

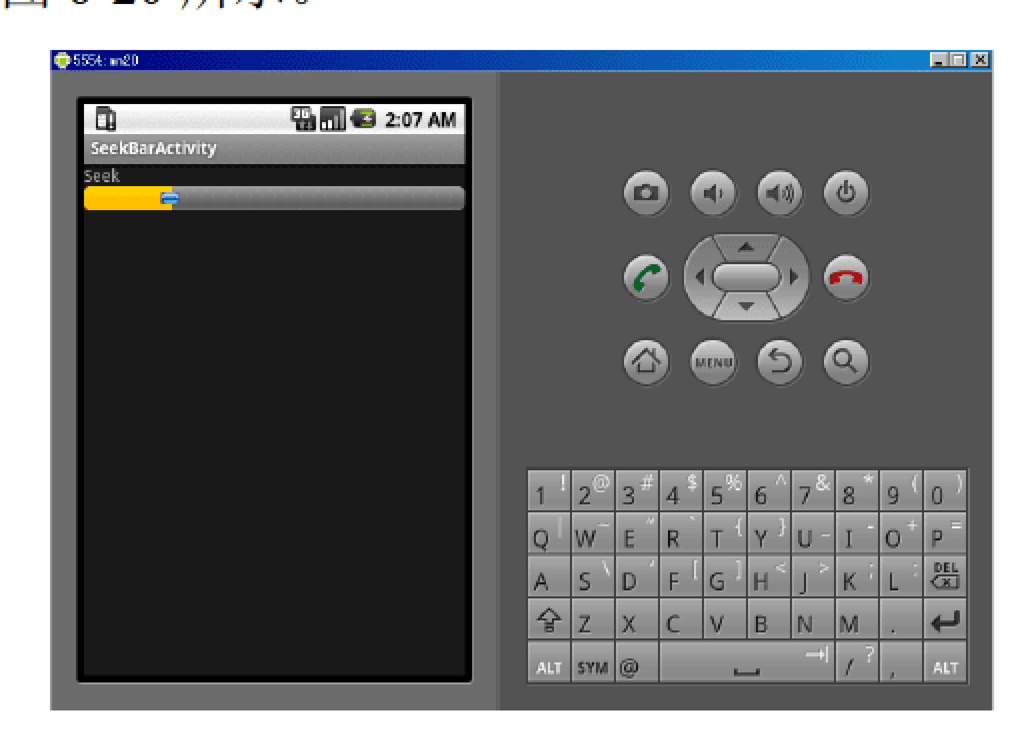


图 6-20　运行效果

6.1.13　自己的分量有几何

评分组件 RatingBar 的功能是为用户提供一个评分操作的模式。在日常应用中，经常见到评分系统，用户可以对某个产品或某个观点进行评分处理。

使用评分组件 RatingBar 的基本操作流程如下所示。

第 1 步：在文件 main.xml 中插入一个按钮，然后为按钮处理事件编写代码。对应的代码如下所示：

```
private    Button.OnClickListener    rating bar button listener    =    new
Button.OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, RatingBarActivity.class);
            startActivity(intent);
        }
};
```

通过上述代码，当用户单击按钮后会跳转到 RatingBarActivity。

第 2 步：创建一个 Activity，为其指定模板 rating_bar.xml，定义一个 RatingBar 控件，设置其 ID 为 rating_bar，并设置宽度和高度都是自适应。

第 3 步：在文件 AndroidManifest.xml 中增加了对 RatingBarActivity 的声明。对应的代码如下所示：

```
<activity Android:name="RatingBarActivity" />
```

至此，整个演练就全部结束了。执行后将显示对应样式的评级图，用户可以通过鼠标来选择评级，运行结果如图 6-21 所示。

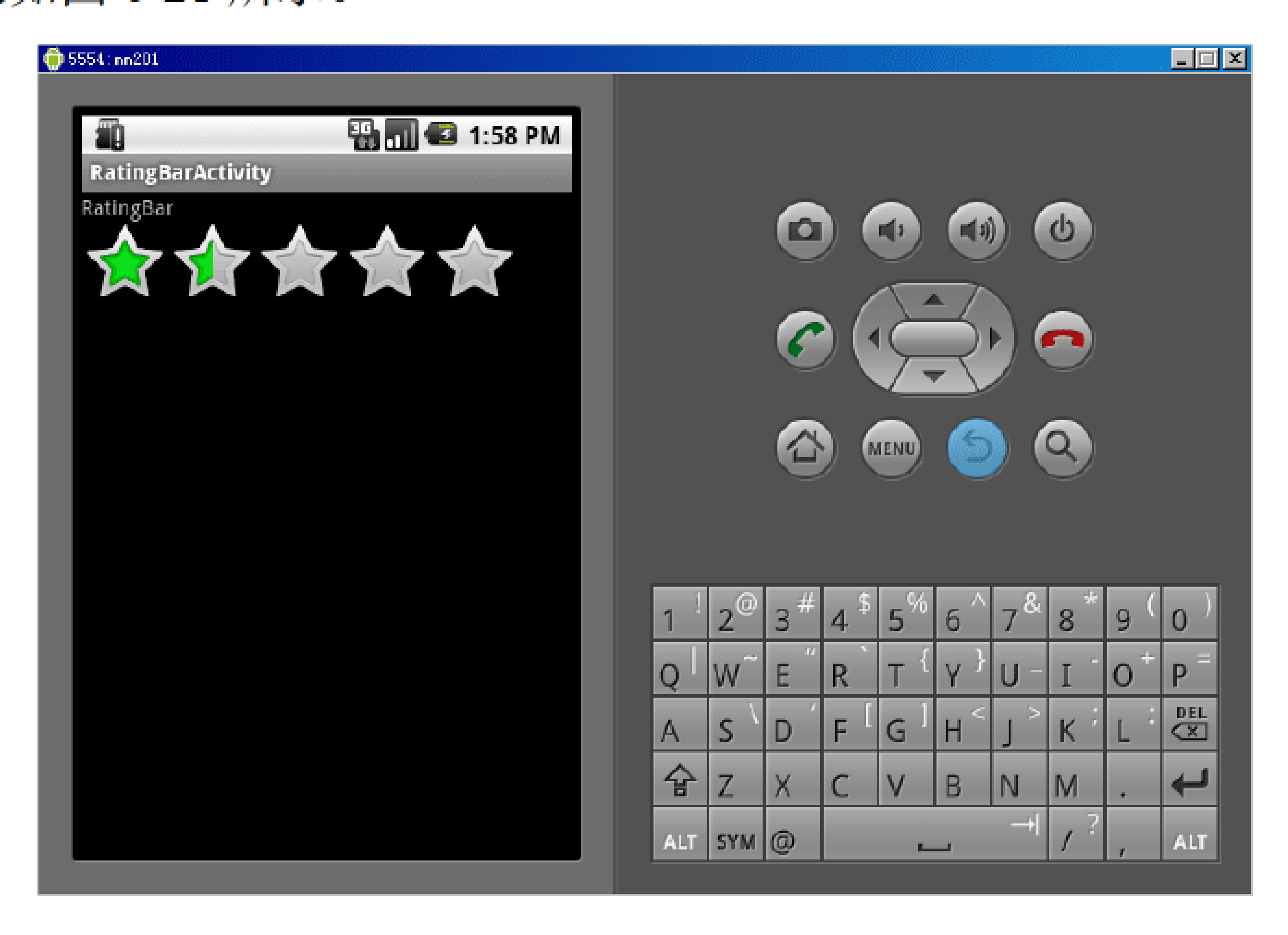


图 6-21　运行效果

6.1.14　图片的绚丽

在安卓应用项目中，可以使用图片视图控件 ImageView 在屏幕中显示一幅图片。

使用图片视图控件 ImageView 的基本操作流程如下所示。

第 1 步：在文件 main.xml 中插入一个按钮，然后为按钮处理事件编写代码，当用户单击按钮后会跳转到 ImageViewActivity。对应代码如下所示：

```
private Button.OnClickListener image_view_button_listener = new Button.
OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, ImageViewActivity.class);
            startActivity(intent);
```

```
        }
};
```

第 2 步：创建一个 Activity，为其指定模板 image_view.xml。文件 mage_view.xml 的具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:orientation="vertical" Android:layout_width="fill_parent"
    Android:layout_height="wrap_content">
<TextView
       Android:layout_width="wrap_content"
       Android:layout_height="wrap_content"
       Android:text="图片展示:"
/>
<ImageView
  Android:id="@+id/imagebutton"
  Android:src="@drawable/eoe"
  Android:layout_width="wrap_content"
  Android:layout_height="wrap_content"/>
</LinearLayout>
```

在上述代码中，设置了 Android:src 为一张图片，该图片位于本项目根目录下的"res\drawable"文件夹中，它支持 PNG、JPG、GIF 等常见的图片格式。

第 3 步：编写对应的 Java 程序。具体代码如下所示：

```
public class ImageViewActivity extends Activity {
    CheckBox plain_cb;
    CheckBox serif_cb;
    CheckBox italic cb;
    CheckBox bold_cb;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("ImageViewActivity");
        setContentView(R.layout.image_view);
    }
}
```

第 4 步：在文件 AndroidManifest.xml 中增加对 ImageViewActivity 的声明。对应代码如下所示：

```
<activity Android:name="ImageViewActivity" />
```

至此，整个演练就全部结束了，执行后将显示对应的图片信息。

6.1.15　图片可以当按钮

安卓中也可以使用图片来作为按钮，秘籍中的描述为：图片按钮控件 ImageButton 的功能

是在系统中将一幅图片作为按钮来使用。通过使用 ImageButton，可以使项目中的按钮更加美观大方。

使用图片按钮控件 ImageButton 的基本操作流程如下。

第 1 步：在文件 main.xml 中插入一个按钮，然后为按钮处理事件编写代码，当用户单击按钮后会跳转到 ImageButtonActivity。对应代码如下所示。

```
private  Button.OnClickListener  image_button_button_listener  =  new  Button.
OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, ImageButtonActivity.class);
            startActivity(intent);
        }
};
```

第 2 步：为创建的 Activity 指定模板 image_button.xml，设置 Android:src 为一张图片，该图片位于本项目根目录下的“res\drawable”文件夹中，它支持 PNG、JPG、GIF 等常见的图片格式。文件 image_button.xml 的具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:orientation="vertical" Android:layout_width="fill_parent"
    Android:layout_height="wrap_content">
<TextView
       Android:layout_width="wrap_content"
       Android:layout height="wrap content"
       Android:text="图片按钮:"
/>
<ImageButton id="@+id/imagebutton"
  Android:src="@drawable/play"
  Android:layout_width="wrap_content"
  Android:layout_height="wrap_content"/>
</LinearLayout>
```

第 3 步：编写对应的 Java 程序。具体代码如下所示：

```
public class ImageButtonActivity extends Activity {
    CheckBox plain_cb;
    CheckBox serif_cb;
    CheckBox italic_cb;
    CheckBox bold cb;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("ImageButtonActivity");
        setContentView(R.layout.image_button);
//      find_and_modify_text_view();
    }
}
```

第 4 步：在文件 AndroidManifest.xml 中增加对 ImageButtonActivity 的声明。对应代码如下所示：

```
<activity Android:name="ImageButtonActivity" />
```

至此，整个演练就全部结束了。执行后将显示一个按钮，此按钮使用指定的图片来实现，运行效果如图 6-22 所示。

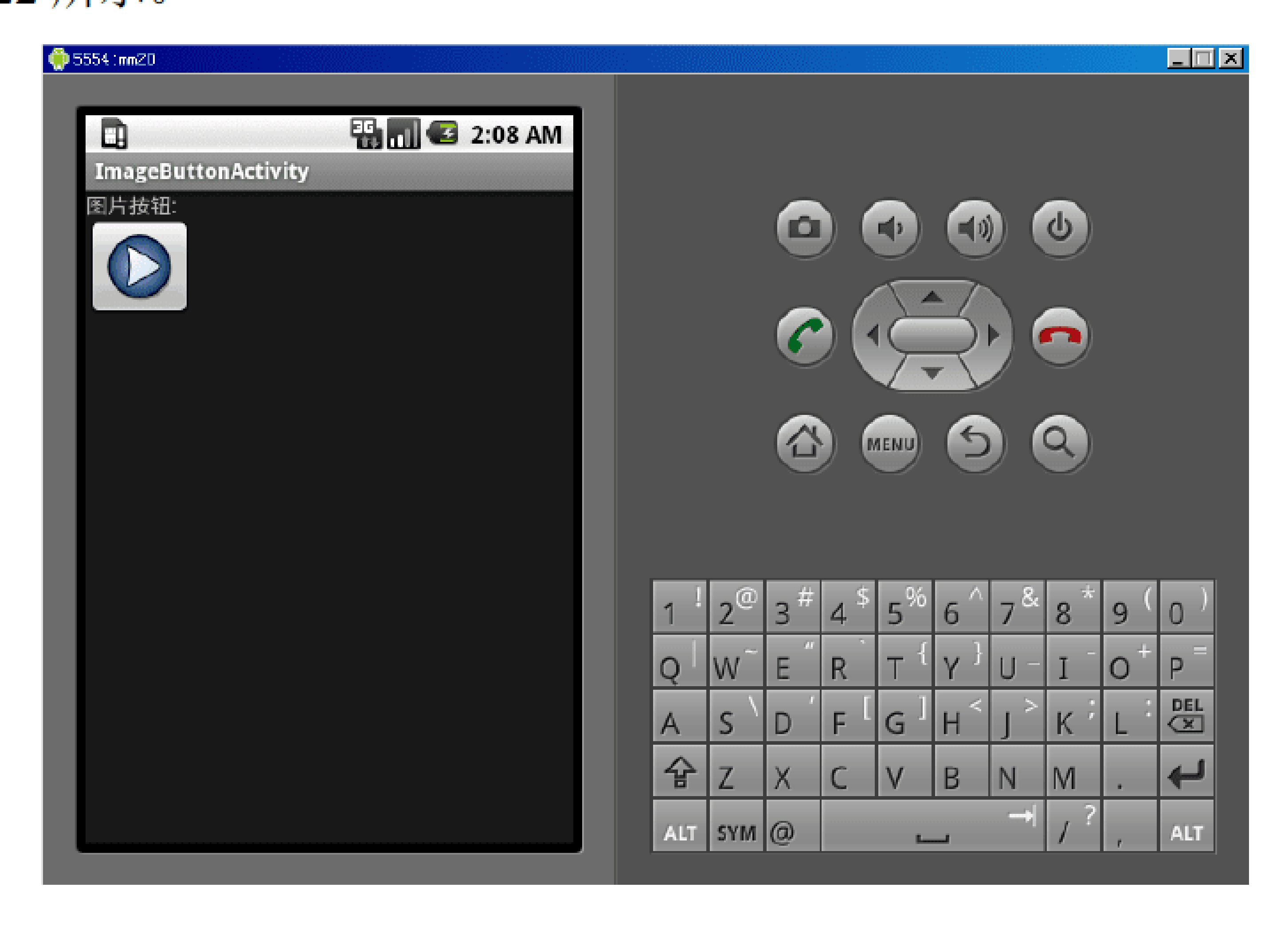


图 6-22　运行效果

6.1.16　追忆往事

在安卓中有两个切换图片的控件，分别是 ImageSwitcher 和 Gallery，它们能够以滑动的方式展现图片。在具体效果上，将首先显示一幅大图，然后在大图下面再显示一组可以滚动的小图。上述显示方式在现实中十分常见，例如 QQ 空间的照片，如图 6-23 所示。

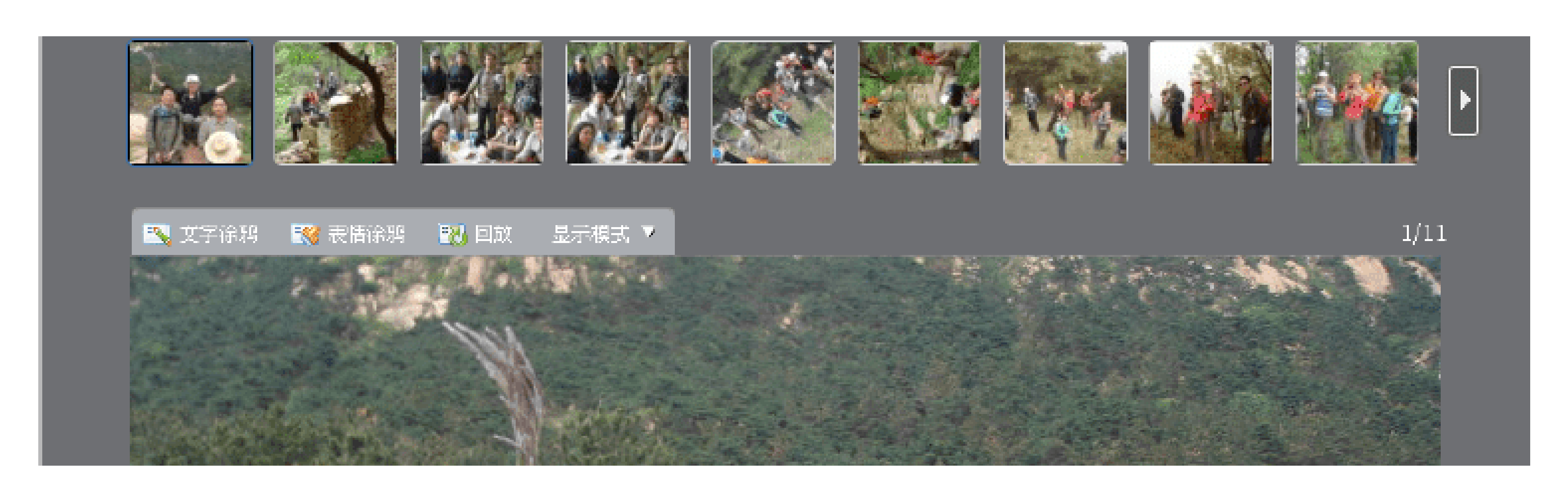


图 6-23　QQ 空间照片

使用切换图片控件 ImageSwitcher 和 Gallery 的基本操作流程如下。

第 1 步：在文件 main.xml 中插入一个按钮，然后为按钮处理事件编写代码，当用户单击按钮后会跳转到 ImageShowActivity。对应代码如下所示：

```
private Button.OnClickListener image_show_button_listener = new Button.
OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, ImageShowActivity.class);
            startActivity(intent);
```

```
        }
};
```

第 2 步：为创建的 Activity 指定模板 image_show.xml，在 RelativeLayout 中插入了 2 个控件，分别是 ImageSwitcher 和 Gallery。其中 ImageSwitcher 用于显示大图，Gallery 用于控制小图列表索引。

第 3 步：编写对应的 Java 程序。首先编写 onCreate 函数，对应的代码如下所示：

```
public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       requestWindowFeature(Window.FEATURE_NO_TITLE);
       setContentView(R.layout.image_show);
       setTitle("ImageShowActivity");
       mSwitcher = (ImageSwitcher) findViewById(R.id.switcher);
       mSwitcher.setFactory(this);
       mSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
       Android.R.anim.fade_in));
       mSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
       Android.R.anim.fade_out));
       Gallery g = (Gallery) findViewById(R.id.gallery);
       g.setAdapter(new ImageAdapter(this));
       g.setOnItemSelectedListener(this);
}
```

在上述代码中，通过使用 requestWindowFeature(Window.FEATURE_NO_TITLE) 设置 Activity 没有 Titlebar，这样，此图片的显示区域就会增大。而类 Gallery 的使用方法和 ListView 差不多，也需要使用 setAdapter 来进行资源设置。

第 4 步：对 BaseAdapter 进行封装，通过 GetView 函数来返回要显示的 ImageView。GetView 函数的具体实现代码如下所示：

```
public View getView(int position, View convertView, ViewGroup parent) {
          ImageView i = new ImageView(mContext);
          i.setImageResource(mThumbIds[position]);
          i.setAdjustViewBounds(true);
          i.setLayoutParams(new Gallery.LayoutParams(
          LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
          i.setBackgroundResource(R.drawable.picture_frame);
          return i;
}
```

在上述代码中，首先动态生成了一个 ImageView，然后使用 setImageResource、setLayoutParams 和 setBackgroundResource 分别实现了图片源文件、图片大小和图片背景的设置。当图片被显示到当前屏幕时，此函数会自动回调来提供要显示的 ImageView。

第 5 步：在 ImageSwitcher1 中实现 ViewSwitcher.ViewFactory 接口，在 ViewSwitcher.ViewFactory 接口中存在方法 View makeView。其实现代码如下所示：

```
public View makeView() {
       ImageView i = new ImageView(this);
       i.setBackgroundColor(0xFF000000);
       i.setScaleType(ImageView.ScaleType.FIT_CENTER);
```

```
        i.setLayoutParams(new ImageSwitcher.LayoutParams
         (LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));
        return i;
}
```

通过上述代码，为 ImageSwitcher 返回了一个 View，在调用 ImageSwitcher 时，首先通过 Factory 为其提供一个 View，然后 ImageSwitcher 就可以初始化各种资源了。

第 6 步：在文件 AndroidManifest.xml 中增加对 Activity 的声明。对应代码如下所示：

```
<activity Android:name="ImageShowActivity" />
```

至此，整个演练就全部结束了。执行后将会按照 QQ 空间中的照片样式显示，执行效果如图 6-24 所示。

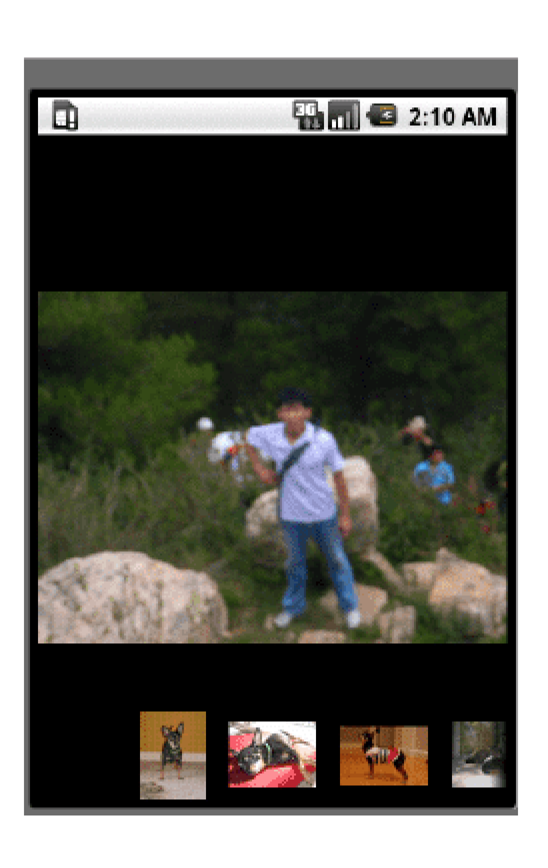


图 6-24　执行效果

6.1.17　网格

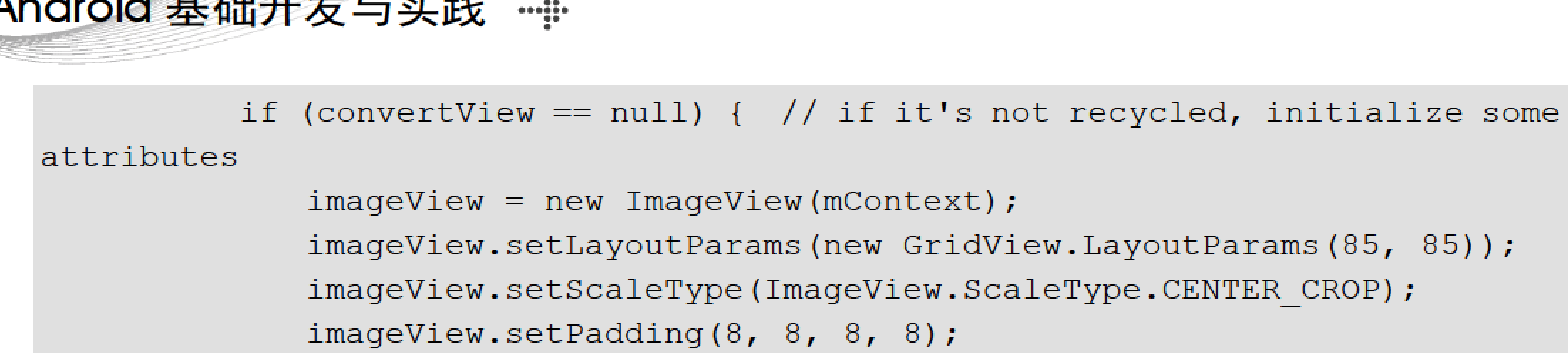
网格视图控件 GridView 的功能是将很多幅指定的图片以指定的大小显示出来。此功能在相册的图片浏览中比较常见。使用网格视图控件 GridView 的基本操作流程如下所示。

第 1 步：编写文件 GridViewActivity.java，首先创建 onCreate 方法，为创建的 Activity 指定模板 grid_view.xml。具体代码如下所示：

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.grid_view);
        setTitle("GridViewActivity");
        GridView gridview = (GridView) findViewById(R.id.grid_view);
        gridview.setAdapter(new ImageAdapter(this));
}
```

第 2 步：获取其模板中的 GridView 控件，并使用 setAdapter 方法为其绑定一个合适的 ImageAdapter，最后编写实现 ImageAdapter 的代码。具体代码如下所示：

```
public class ImageAdapter extends BaseAdapter {
        private Context mContext;
        public ImageAdapter(Context c) {
            mContext = c;
        }
        public int getCount() {
            return mThumbIds.length;
        }
        public Object getItem(int position) {
            return null;
        }
        public long getItemId(int position) {
            return 0;
        }
        public View getView(int position, View convertView, ViewGroup parent) {
            ImageView imageView;
```



```
            if (convertView == null) {  // if it's not recycled, initialize some
attributes
                imageView = new ImageView(mContext);
                imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
                imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
                imageView.setPadding(8, 8, 8, 8);
            } else {
                imageView = (ImageView) convertView;
            }
            imageView.setImageResource(mThumbIds[position]);
            return imageView;
        }
        // references to our images
        private Integer[] mThumbIds = {
                R.drawable.grid_view_01, R.drawable.grid_view_02,
                R.drawable.grid_view_03, R.drawable.grid_view_04,
                R.drawable.grid_view_05, R.drawable.grid_view_06,
                R.drawable.grid_view_07, R.drawable.grid_view_08,
                R.drawable.grid_view_09, R.drawable.grid_view_10,
                R.drawable.grid_view_11, R.drawable.grid_view_12,
                R.drawable.grid_view_13, R.drawable.grid_view_14,
                R.drawable.grid_view_15, R.drawable.sample_1,
                R.drawable.sample_2, R.drawable.sample_3,
                R.drawable.sample_4, R.drawable.sample_5,
                R.drawable.sample_6, R.drawable.sample_7
        };
}
```

在上述代码中，因为 ImageAdapter 继承于 BaseAdapter，所以通过构造方法 ImageAdapter 获取 Context，然后实现了 getView。

第 3 步：在文件 AndroidManifest.xml 中增加对 Activity 的声明。对应代码如下所示：

```
<activity Android:name="GridViewActivity" />
```

至此，整个演练就成功完成了。执行后将会按照网格视图的方式显示指定的图片，运行效果如图 6-25 所示。



图 6-25　运行效果

6.2　欲穷千里目，更上一层楼

“欲穷千里目，更上一层楼”，安卓组件的功能博大精深，要想使功力更上一层楼，还需要继续专心修炼组件，把基础打牢固，才能达到至高境界。前面只是介绍了 Android 中主要组件的使用方法，其实在 Android 中还有很多其他组件，而且利用这些组件能够实现更加复杂的功能。

6.2.1　在对话框中使用进度条

秘籍中说道：ProgressDialog 进度条的功能是在对话框中实现显示进度条的效果。

题　目	目　的	源码路径
演练 2	ProgressDialog 的基本使用方法	“光盘\daima\6\jindutiao”文件夹

第 1 步：编写布局文件 main.xml，用于在界面中插入 2 个 Button 按钮。

第 2 步：编写文件 strings.xml，用于设置标题文本。其具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">对话框进度条</string>
   <string name="app_name">对话框进度条</string>
</resources>
```

第 3 步：编写文件 Activity01.java，实现单击 Button 按钮后的事件处理程序。其具体实现流程如下。

(1)　声明进度条对话框 m_pDialog，然后得到按钮对象 mButton01 和 mButton02。

(2)　设置 mButton01 的事件监听，首先创建 ProgressDialog 对象，然后设置进度条风格，风格为圆形、旋转的，最后分别设置 ProgressDialog 标题、提示信息、图标等信息。

(3)　设置 mButton02 的事件监听，首先创建 ProgressDialog 对象，然后设置进度条风格，风格为长形，最后分别设置 ProgressDialog 标题、提示信息、图标等信息。

(4)　显示 ProgressDialog 中的内容。

文件 Activity01.java 的主要实现代码如下所示：

```
public class Activity01 extends Activity
{
    private Button mButton01,mButton02;
    int m_count = 0;
    //声明进度条对话框
    ProgressDialog m_pDialog;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //得到按钮对象
```

```
        mButton01 = (Button)findViewById(R.id.Button01);
        mButton02 = (Button)findViewById(R.id.Button02);
        //设置 mButton01 的事件监听
        mButton01.setOnClickListener(new Button.OnClickListener() {
            @Override
            public void onClick(View v)
            {
                // TODO Auto-generated method stub
                //创建 ProgressDialog 对象
                m_pDialog = new ProgressDialog(Activity01.this);
                // 设置进度条风格，风格为圆形、旋转的
                m_pDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
                // 设置 ProgressDialog 标题
                m_pDialog.setTitle("提示");

                // 设置 ProgressDialog 提示信息
                m_pDialog.setMessage("圆形进度条对话框");
                // 设置 ProgressDialog 标题图标
                m_pDialog.setIcon(R.drawable.img1);
                // 设置 ProgressDialog 的进度条是否不明确
                m_pDialog.setIndeterminate(false);

                // 设置 ProgressDialog 是否可以按退回按键取消
                m_pDialog.setCancelable(true);

                // 设置 ProgressDialog 的一个 Button
                m_pDialog.setButton("确定", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int i)
                    {
                        //点击“确定”按钮取消对话框
                        dialog.cancel();
                    }
                });
                // 让 ProgressDialog 显示
                m_pDialog.show();
            }
        });

      //设置 mButton02 的事件监听
        mButton02.setOnClickListener(new Button.OnClickListener() {
            @Override
            public void onClick(View v)
            {
                // TODO Auto-generated method stub
                m_count = 0;
                // 创建 ProgressDialog 对象
                m_pDialog = new ProgressDialog(Activity01.this);
                // 设置进度条风格，风格为长形
                m_pDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
                // 设置 ProgressDialog 标题
                m_pDialog.setTitle("提示");
                // 设置 ProgressDialog 提示信息
                m_pDialog.setMessage("长形对话框进度条");
                // 设置 ProgressDialog 标题图标
```

```
                m_pDialog.setIcon(R.drawable.img2);
                // 设置 ProgressDialog 进度条进度
                m pDialog.setProgress(100);
                // 设置 ProgressDialog 的进度条是否不明确
                m_pDialog.setIndeterminate(false);
                                // 设置 ProgressDialog 是否可以按退回按键取消
                m_pDialog.setCancelable(true);
                                // 让 ProgressDialog 显示
                m_pDialog.show();
                                new Thread()
                {
                    public void run()
                    {
                        try
                        {
                            while (m_count <= 100)
                            {
                                // 由线程来控制进度
                                m_pDialog.setProgress(m_count++);
                                Thread.sleep(100);
                            }
                            m_pDialog.cancel();
                        }
                        catch (InterruptedException e)
                        {
                            m_pDialog.cancel();
                        }
                    }
                }.start();

            }
        });
    }
}
```

至此，整个演练就完美结束了，执行后的效果如图 6-26 所示。

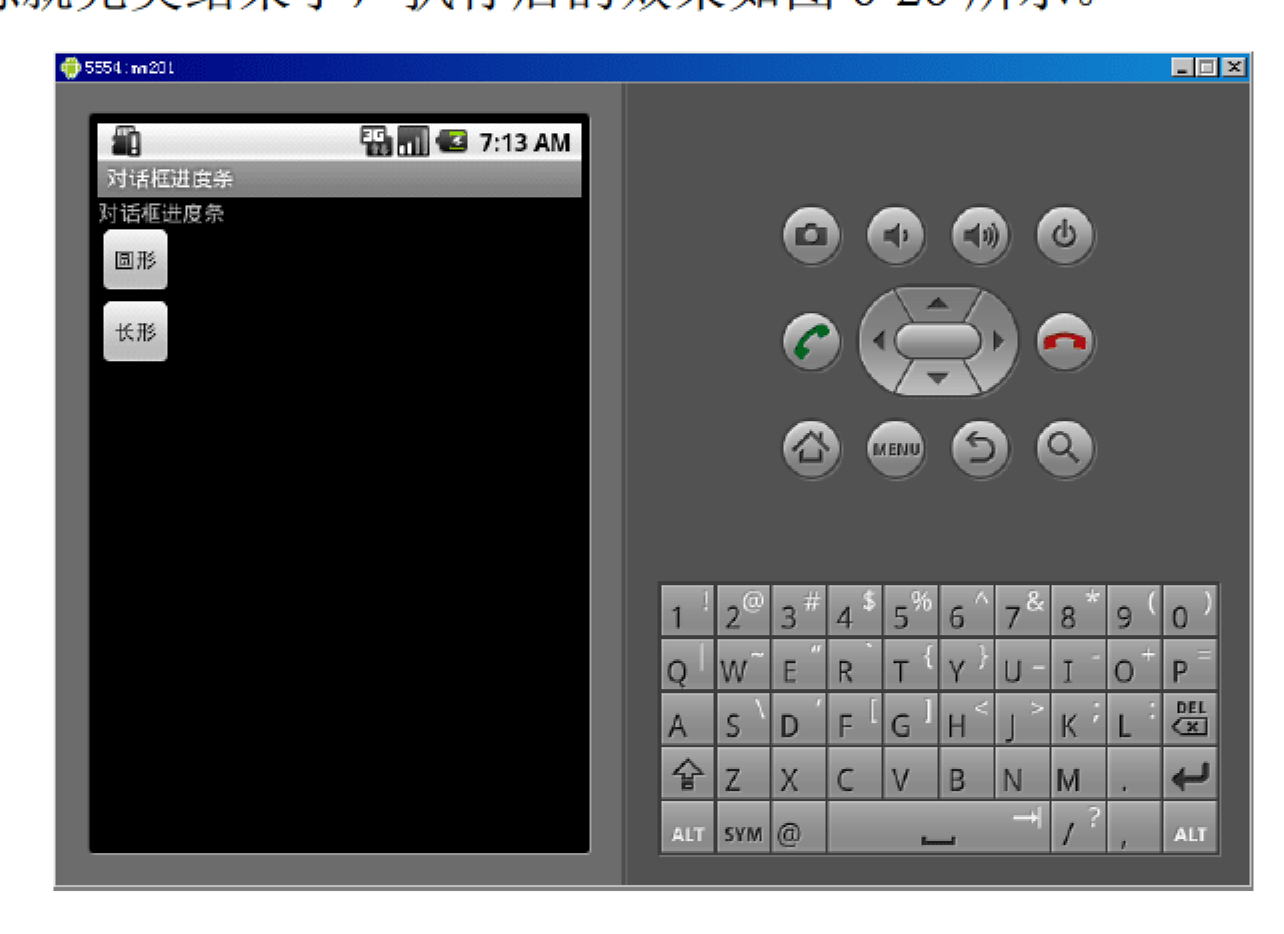


图 6-26　执行效果

当单击图 6-27 中的“圆形”进度条按钮后，将显示圆形化的进度条效果，如图 6-27 所示；

当单击图 6-27 中的“长形”进度条按钮后，将显示长形的进度条效果，如图 6-28 所示。




图 6-27　圆形化的进度条



图 6-28　长形化的进度条

6.2.2　使用 Spinner 和 setDropDownViewResource

题　目	目　的	源码路径
演练 3	使用 Spinner 和 setDropDownViewResource 的基本方法	“光盘：\daima\6\Spinnergaoji”文件夹

注意： 本演练使用了 ArrayAdapter(Context context、int textViewResourceId、T[] objects)这个 Constructor，其中 textViewResourceId 使用 Android 提供的 ResourceID，objects 为必须传递的字符串数组(String Array)。Adapter 的 setDropDownViewResource 可以设置下拉菜单的显示方式，将该 xml 定义在 res/layout 目录下面，可针对下拉菜单中的 TextView 进行设置，如同本程序里的 R.layout.myspinner_dropdown 即为自定义的下拉菜单 TextView 样式。除了改变下拉菜单样式外，也对 Spinner 做了一点动态效果，单击 Spinner 时，晃动 Spinner 再出现下拉菜单(myAnimation)。

本演练的具体实现流程如下。

第 1 步：编写布局文件 main.xml，插入一个 Button 和一个 TextView。

第 2 步：编写 color.xml，设置界面的颜色。具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <drawable name="black">#000000</drawable>
  <drawable name="white">#FFFFFFFF</drawable>
</resources>
```

第 3 步：在 res 目录下新建一个 anim 文件夹，用来存放动画效果，在其中新建一个 my_anim.xml 文件。

第 4 步：在 res 目录下的 layout 文件夹中新建一个 myspinner_dropdown.xml 文件，用来存放下拉菜单弹出内容的布局。

第 5 步：编写事件处理文件 Spinnergaoji.java，其具体实现流程如下。

(1)　定义一个下拉菜单，以 findViewById()取得对象。

(2)　定义一个字符串数组和一个 ArrayAdepter，用于显示供选择的国家。

(3)　给下拉菜单内容分别设置样式、内容适配器，并添加动画。

文件 Spinnergaoji.java 的主要实现代码如下所示：

```
public class Spinnergaoji extends Activity
{
  private static final String[] countriesStr =
  { "AA", "BB", "CC", "DD" };
  private TextView myTextView;
  private Spinner mySpinner;
  private ArrayAdapter<String> adapter;
  Animation myAnimation;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    /* 载入 main.xml Layout */
    setContentView(R.layout.main);
    /* 以 findViewById()取得对象 */
    myTextView = (TextView) findViewById(R.id.myTextView);
    mySpinner = (Spinner) findViewById(R.id.mySpinner);
    adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_item, countriesStr);
    /* myspinner_dropdown 为自定义下拉菜单模式定义在 res/layout 目录下 */
    adapter.setDropDownViewResource(R.layout.myspinner dropdown);
    /* 将 ArrayAdapter 添加 Spinner 对象中 */
    mySpinner.setAdapter(adapter);
    /* 将 mySpinner 添加 OnItemSelectedListener */
    mySpinner.setOnItemSelectedListener
     (new Spinner.OnItemSelectedListener()
    {
      @Override
      public void onItemSelected
       (AdapterView<?> arg0, View arg1, int arg2,
          long arg3)
      {
        /* 将所选 mySpinner 的值带入 myTextView 中 */
        myTextView.setText("您选择的是" + countriesStr[arg2]);
        /* 将 mySpinner 显示 */
        arg0.setVisibility(View.VISIBLE);
      }
      @Override
      public void onNothingSelected(AdapterView<?> arg0)
      {
        // TODO Auto-generated method stub
      }
    });
    /* 取得 Animation 定义在 res/anim 目录下 */
```

```
    myAnimation = AnimationUtils.loadAnimation(this, R.anim.my anim);
    /* 将 mySpinner 添加 OnTouchListener */
    mySpinner.setOnTouchListener(new Spinner.OnTouchListener()
    {
      @Override
      public boolean onTouch(View v, MotionEvent event)
      {
        /* 将 mySpinner 运行 Animation */
        v.startAnimation(myAnimation);
        /* 将 mySpinner 隐藏 */
        v.setVisibility(View.INVISIBLE);
        return false;
      }
    });
    mySpinner.setOnFocusChangeListener(new Spinner.OnFocusChangeListener()
    {
      @Override
      public void onFocusChange(View v, boolean hasFocus)
      {
        // TODO Auto-generated method stub
      }
    });
  }
}
```

至此，整个演练就结束了。执行后的效果如图 6-29 所示，当单击下拉框时会弹出一个浮动的可选的选项框，在此用户可以选择一个选项，如图 6-30 所示。



图 6-29　执行效果

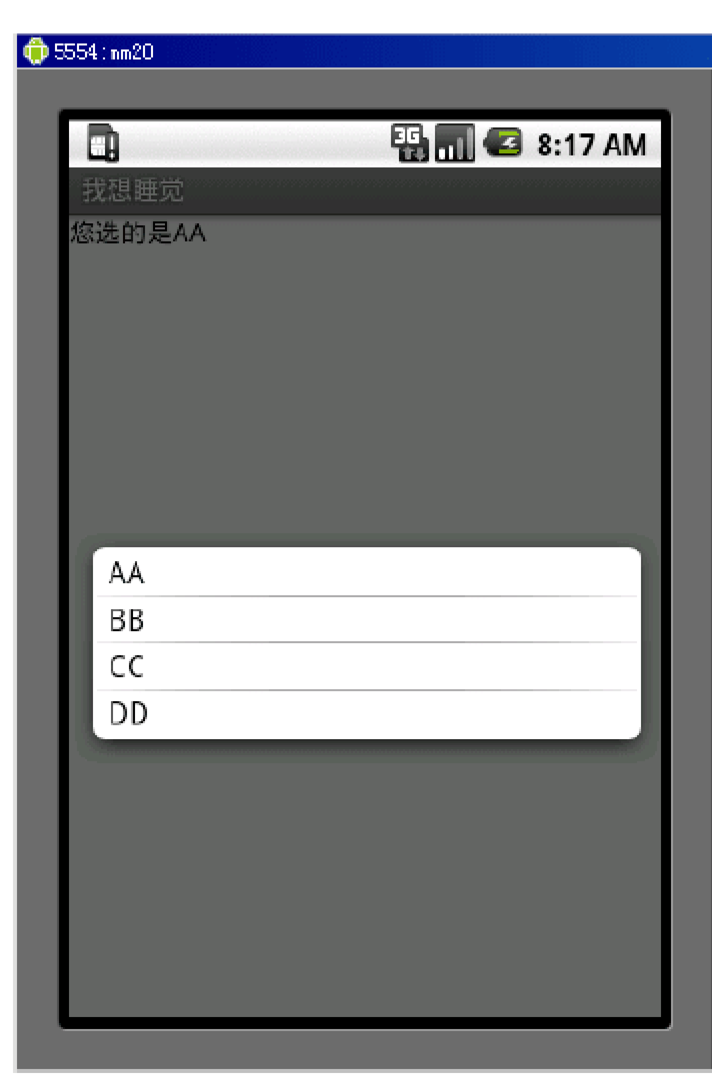


图 6-30　显示浮动选项框

Animation 主要有两种动态方式，一种是 tweened animation(渐变动画)；另一种是 frame by frame animation(画面转换动画)。tweened animation 主要有以下 4 种基本转换方式。

(1) AlphaAnimation (transparency changes)：透明度转换。

(2) RotateAnimation (rotations)：旋转转换。

(3)　ScaleAnimation (growing or shrinking)：缩放转换。

(4)　TranslateAnimation (position changes)：位置转换。

定义好你想要的动画 xml 后，用 AnimationUtils.loadAnimation 将动画加载，在想要加上动态效果的组件中使用 startAnimation 方法。

6.2.3　Gallery 和 BaseAdapter 容器

题　目	目　的	源码路径
演练 4	实战演练 Gallery 和 BaseAdapter 联合使用的方法	“光盘：\daima\6\UseGallery”文件夹

注意： 前面已经学习了 Gallery 的方法，现在将数张 PNG 图片导入 Drawable 中，并在 onCreate 时载入到 Gallery Widget 中，然后试着再添加一个 OnItemClick 的事件，以获取图片的 ID 编号来响应用户单击图片时的状态，完成 Gallery 的高级使用。本演练的重点是如何设置 Gallery 图片的宽度、高度以及放置图片 Layout 的大小，在此改写一个继承自 BaseAdapter 的 ImageAdapter 容器来存放图片，通过 ImageView.setScaleType() 的方法来改变图片的显示，再通过 setLayoutParams() 方法来改变 Layout 的宽度和高度。

本实例的具体实现流程如下。

第 1 步：编写布局文件 main.xml，添加一个 Gallery 和一个 ImageView。

第 2 步：定义 layout 外部 resource 的 xml 文件，用来改变 layout 的背景。

第 3 步：新建一个 myImageAdapter 类——Gallery 的适配器，它继承于 BaseAdapter 类。具体代码如下所示：

```
public class myImageAdapter extends BaseAdapter {
    @Override
    public int getCount() {
        // TODO Auto-generated method stub
        return 0;
    }
    @Override
    public Object getItem(int position) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public long getItemId(int position) {
        // TODO Auto-generated method stub
        return 0;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // TODO Auto-generated method stub
        return null;
    }
}
```

第 4 步：修改 mainActivity.java，添加 Gallery 相关操作。主要代码如下所示：

```
public class Galleryjia extends Activity
```

```
{
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /*通过 findViewById 取得*/
    Gallery g = (Gallery) findViewById(R.id.mygallery);
    /* 添加一 ImageAdapter 并设置给 Gallery 对象 */
    g.setAdapter(new ImageAdapter(this));

    /* 设置一个 itemclickListener 并 Toast 被单击图片的位置 */
    g.setOnItemClickListener(new OnItemClickListener()
    {
      public void onItemClick
      (AdapterView<?> parent, View v, int position, long id)
      {
        Toast.makeText
        (Galleryjia.this, getString(R.string.my_gallery_text_pre)
        + position+ getString(R.string.my_gallery_text_post),
        Toast.LENGTH_SHORT).show();
      }
    });
  }

  /* 改写 BaseAdapter 自定义一 ImageAdapter class */
  public class ImageAdapter extends BaseAdapter
  {
    /*声明变量*/
    int mGalleryItemBackground;
    private Context mContext;
       /*ImageAdapter 的构造器*/
    public ImageAdapter(Context c)
    {
      mContext = c;
      /* 使用在 res/values/attrs.xml 中的<declare-styleable>定义
      * 的 Gallery 属性.*/
      TypedArray a = obtainStyledAttributes(R.styleable.Gallery);
      /*取得 Gallery 属性的 Index id*/
      mGalleryItemBackground = a.getResourceId
      (R.styleable.Gallery_android_galleryItemBackground, 0);

      /*让对象的 styleable 属性能够反复使用*/
      a.recycle();
    }
    /* 覆盖的方法 getCount，返回图片数目 */
    public int getCount()
    {
      return myImageIds.length;
    }
    /* 覆盖的方法 getItemId，返回图像的数组 id */
    public Object getItem(int position)
    {
```

```
        return position;
      }
      public long getItemId(int position)
      {
        return position;
      }
       /* 覆盖的方法 getView，返回一 View 对象 */
      public View getView
      (int position, View convertView, ViewGroup parent)
      {
        /*产生 ImageView 对象*/
        ImageView i = new ImageView(mContext);
        /*设置图片给 imageView 对象*/
        i.setImageResource(myImageIds[position]);
        /*重新设置图片的宽高*/
        i.setScaleType(ImageView.ScaleType.FIT_XY);
        /*重新设置 Layout 的宽高*/
        i.setLayoutParams(new Gallery.LayoutParams(136, 88));
        /*设置 Gallery 背景图*/
        i.setBackgroundResource(mGalleryItemBackground);
        /*返回 imageView 对象*/
        return i;
      }
      /*建构一 Integer array 并取得预加载 Drawable 的图片 id*/
      private Integer[] myImageIds =
      {
        R.drawable.photo1,
        R.drawable.photo2,
        R.drawable.photo3,
        R.drawable.photo4,
        R.drawable.photo5,
        R.drawable.photo6,
      };
    }
}
```

至此，整个演练就结束了。执行后会实现相册效果，如图 6-31 所示；当选择一幅图片后，此图片会放大显示，并显示此图片的标号。

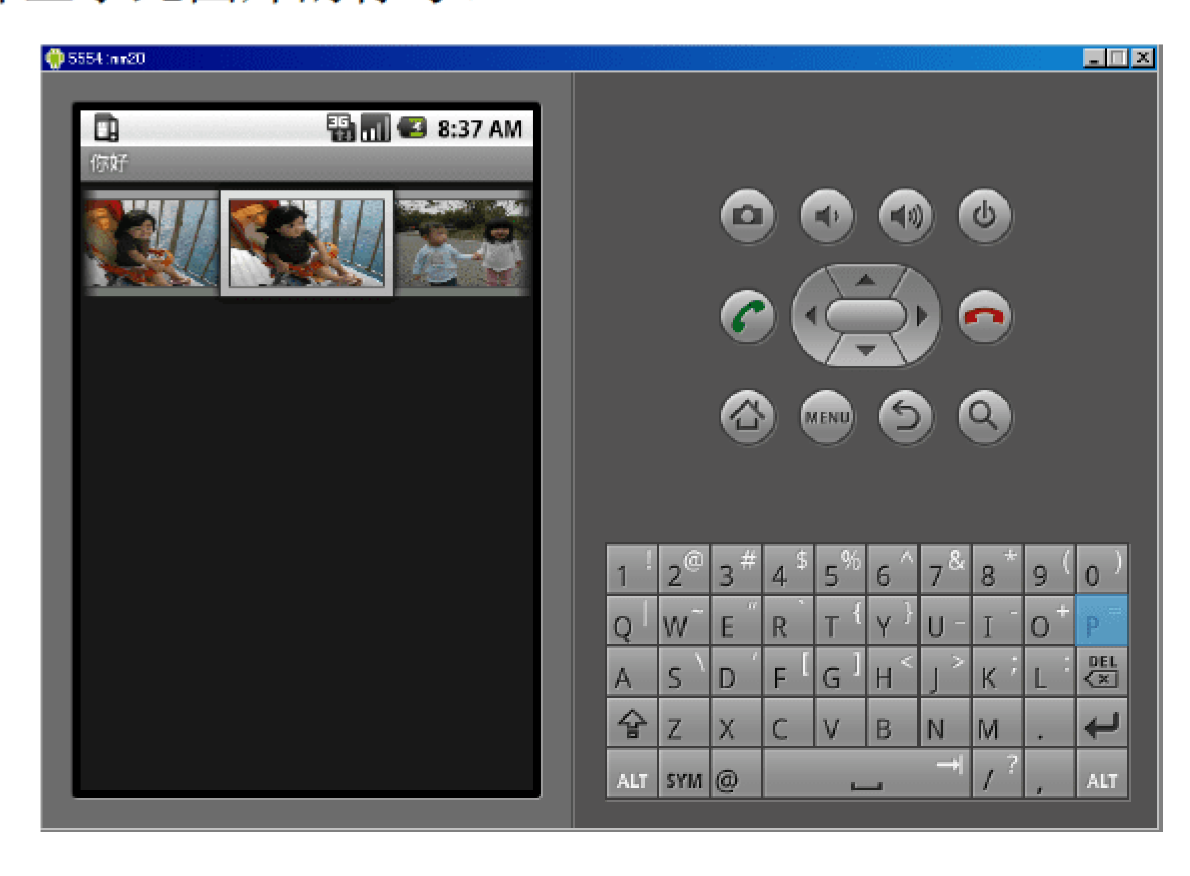


图 6-31　执行效果

6.2.4 实现模拟时钟效果

题　目	目　的	源码路径
演练 5	实战演练用 AnalogClock 和 DigitalClock 实现模拟小时钟的基本流程	“光盘：\daima\6\clock”文件夹

注意： Android 里的 AnalogClock Widget 是一个时钟对象，本实例将配置一个小时钟，并在其下放置一个 TextView。为了做对照，上面放置的为模拟时钟，下面的 TextView 则模拟电子时钟，将 AnalogClock 的时间以数字钟形式显示。本演练的难点是：android.os.Handler、java.lang.Thread 以及 android.os.Message 三对象的整合应用，通过产生 Thread 对象，在进程内同步调用 System.currentTimeMillis() 取得系统时间，并通过 Message 对象来通知 Handler 对象，Handler 则扮演联系 Activity 与 Thread 之间的桥梁，在收到 Message 对象后，将时间变量的值显示于 TextView 中，产生数字时钟的外观与功能。

本实例的具体实现流程如下。

第 1 步：编写布局文件 main.xml，分别添加一个 AnalogClock、一个 DigitalClock、一个 TextView，实现整体布局。

第 2 步：实现模拟时钟效果不需要额外的代码，只需要在 UI 中添加 AnalogClock 控件后即可自动显示。具体代码如下所示：

```
/*定义模拟时钟对象*/
AnalogClock myClock;
/*从 XML 中获取模拟时钟 UI 对象*/
myClock=(AnalogClock)findViewById(R.id.Clock);
```

第 3 步：数字时钟的实现也不需要额外代码，只需在 UI 中添加 DigitalClock 控件，其会自动显示时间。具体代码如下所示：

```
/*定义数字时钟对象*/
DigitalClock myDigClock;
/*从 XML 中获取数字时钟 UI 对象*/
myDigClock=(DigitalClock)findViewById(R.id.DigitalClock01);
```

使用线程实现的 TextView 时钟则需要线程 Thread、Handler(发送、处理消息)辅助实现，具体代码如下所示：

```
import android.os.Handler;
import android.os.Message;
public class Clock extends Activity implements Runnable{
    public Handler myHandler;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.gh);
        myHandler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
```

```
                // TODO Auto-generated method stub
            }
        };
        Thread myT = new Thread(this);
        myT.start();
    }
    @Override
    public void run() {
        // TODO Auto-generated method stub
    }
}
```

第 4 步：实现最终的主程序 shizhong.java，因为要另外加载 Java 的 Calendar 与 Thread 对象，所以需要在 onCreate()方法中构造 Handler 与 Thread 两个对象，并实现 handelMessage()与 run()两个方法。主要实现代码如下所示：

```
/*这里我们需要使用 Handler 类与 Message 类来处理运行线程*/
import android.os.Handler;
……………………………………………
/*需要使用 Java 的 Calendar 与 Thread 类来取得系统时间*/
import java.util.Calendar;
import java.lang.Thread;

public class shizhong extends Activity
{
  /*声明一常数作为判别信息用*/
  protected static final int GUINOTIFIER = 0x1234;

  /*声明两个 widget 对象变量*/
  private TextView mTextView;
  public AnalogClock mAnalogClock;

  /*声明与时间相关的变量*/
  public Calendar mCalendar;
  public int mMinutes;
  public int mHour;

  /*声明 Handler 与 Thread 变量*/
  public Handler mHandler;
  private Thread mClockThread;
  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    /*通过 findViewById 取得两个 widget 对象*/
    mTextView=(TextView)findViewById(R.id.myTextView);
    mAnalogClock=(AnalogClock)findViewById(R.id.myAnalogClock);

    /*通过 Handler 来接收运行线程所传递的信息并更新 TextView*/
    mHandler = new Handler()
```

```
    {
      public void handleMessage(Message msg)
      {
        /*这里是处理信息的方法*/
        switch (msg.what)
        {
          case shizhong.GUINOTIFIER:
          /* 在这处理要 TextView 对象 Show 时间的事件 */
            mTextView.setText(mHour+" : "+mMinutes);
            break;
        }
        super.handleMessage(msg);
      }
    };

    /*通过运行线程来持续取得系统时间*/
     mClockThread=new LooperThread();
     mClockThread.start();
  }

  /*改写一个 Thread Class 用来持续取得系统时间*/
  class LooperThread extends Thread
  {
    public void run()
    {
      super.run();
      try
      {
        do
        {
          /*取得系统时间*/
          long time = System.currentTimeMillis();
          /*通过 Calendar 对象来取得小时与分钟*/
          final Calendar mCalendar = Calendar.getInstance();
          mCalendar.setTimeInMillis(time);
          mHour = mCalendar.get(Calendar.HOUR);
          mMinutes = mCalendar.get(Calendar.MINUTE);

          /*让运行线程休息一秒*/
          Thread.sleep(1000);
           /*关键程序:取得时间后发出信息给 Handler*/
           Message m = new Message();
           m.what = shizhong.GUINOTIFIER;
           shizhong.this.mHandler.sendMessage(m);
        }while(shizhong.LooperThread.interrupted()==false);
        /*当系统发出中断信息时停止本循环*/
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
    }
```

```
    }
}
```

至此，整个演练就结束了。执行后会显示一个时钟效果，具体效果如图 6-32 所示。

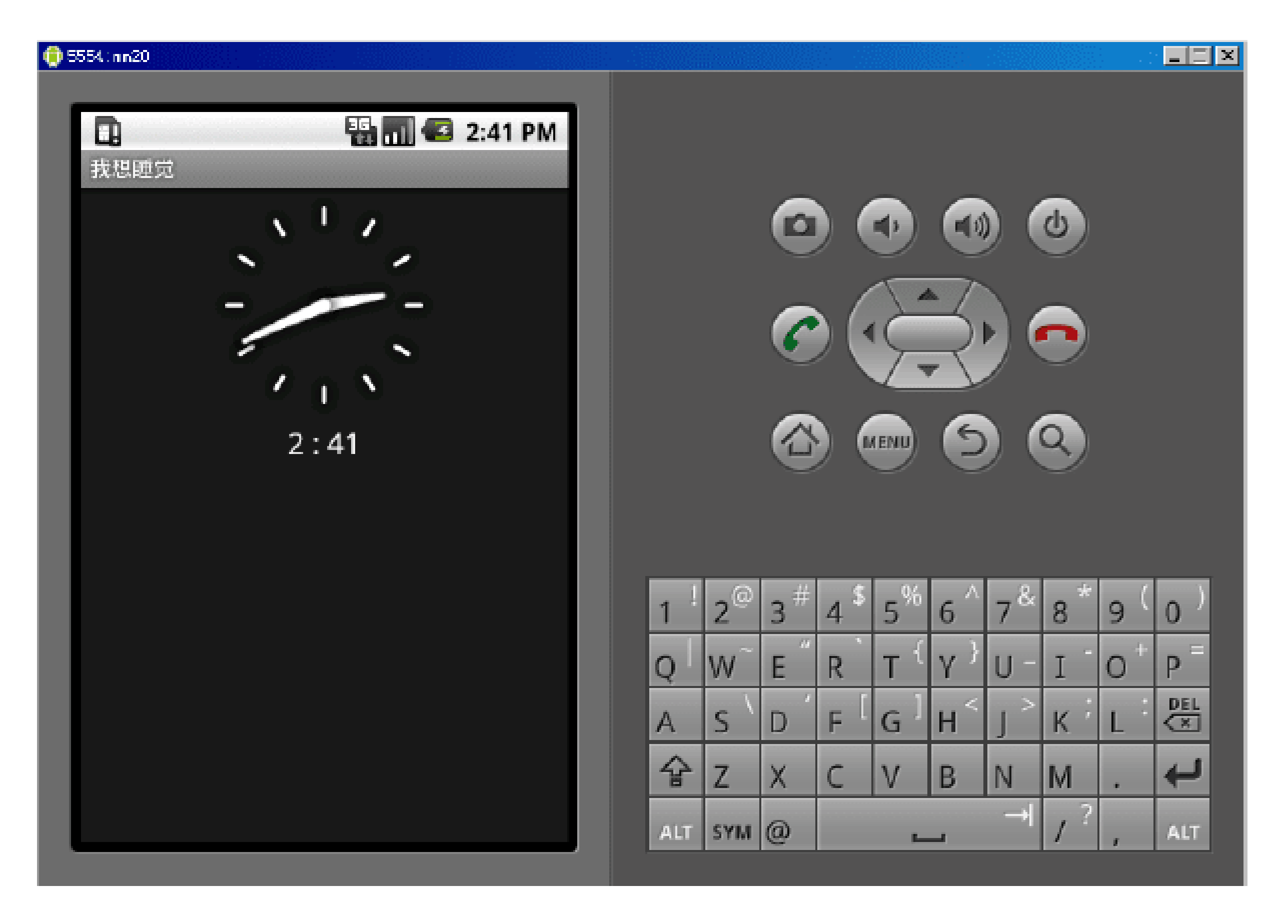


图 6-32　运行效果

6.2.5　FileSearch 文件搜索引擎

秘籍中说道：文件搜索功能在操作系统中或网页中经常遇到，它可以快速帮我们找到想要的文件或资料。如果要在手机中制作一个文件搜索的功能，又该如何实现呢？其实这个功能并不困难，通过 Java I/O 的 API 中提供的 java.io.File 对象，只要利用 File 对象的方法，再搭配 Android 的 EditText、TextView 等对象，就可以轻松做出一个手机的文件搜索引擎。

题　目	目　的	源码路径
演练 6	实战演练 java.io.file 对象的基本使用方法	“光盘:\daima\6\FileSearch”文件夹

注意：演练中将使用 EditText、Button 与 TextView 三种对象来实现此功能，可以将要搜索的文件名称或关键字输入 EditText 中，单击 Button 后，程序会在根目录中寻找符合的文件，并将搜索结果显示于 TextView 中；如果找不到符合的文件，则显示找不到文件。

本实例的具体实现流程如下。

第 1 步：编写布局文件 main.xml，分别添加两个 TextView、一个 EditText 和一个 Button。

第 2 步：在 strings.xml 中添加程序中要使用的字符串。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
  <resources>
    <string name="hello">长夜</string>
    <string name="app_name">漫漫</string>
    <string name="str_button">搜索引擎</string>
    <string name="str_title">输入关键字：</string>
  </resources>
```

第 3 步：编写 FileSearch.java 文件实现搜索功能，其具体实现流程如下。

(1)　声明对象变量并载入 main.xml Layout。然后初始化对象，并为按钮 mButton 添加

onClickListener 单击事件。

(2) 取得输入的关键字，并根据搜索文件的 method 和关键字进行搜索。

文件 FileSearch.java 的主要实现代码如下所示：

```
public class FileSearch extends Activity
{
  /*声明对象变量*/
  private Button mButton;
  private EditText mKeyword;
  private TextView mResult;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    /* 载入 main.xml Layout */
    setContentView(R.layout.main);

    /* 初始化对象 */
    mKeyword=(EditText)findViewById(R.id.mKeyword);
    mButton=(Button)findViewById(R.id.mButton);
    mResult=(TextView) findViewById(R.id.mResult);

    /* 将 mButton 添加 onClickListener */
    mButton.setOnClickListener(new Button.OnClickListener()
    {
      public void onClick(View v)
      {
      /*取得输入的关键字*/
        String keyword = mKeyword.getText().toString();
        if(keyword.equals(""))
        {
          mResult.setText("请勿输入空白的关键字!!");
        }
        else
        {
          mResult.setText(searchFile(keyword));
        }
      }
    });
  }

  /* 搜索文件的 method */
  private String searchFile(String keyword)
  {
    String result="";
    File[] files=new File("/").listFiles();
    for( File f : files )
    {
      if(f.getName().indexOf(keyword)>=0)
      {
```

```
        result+=f.getPath()+"\n";
      }
    }
    if(result.equals("")) result="找不到文件!!";
    return result;
  }
}
```

至此，整个演练就结束了。执行后的初始效果如图 6-33 所示；当输入关键字并单击“搜索引擎”按钮后会检索出对应的信息，如图 6-34 所示。




图 6-33　初始效果

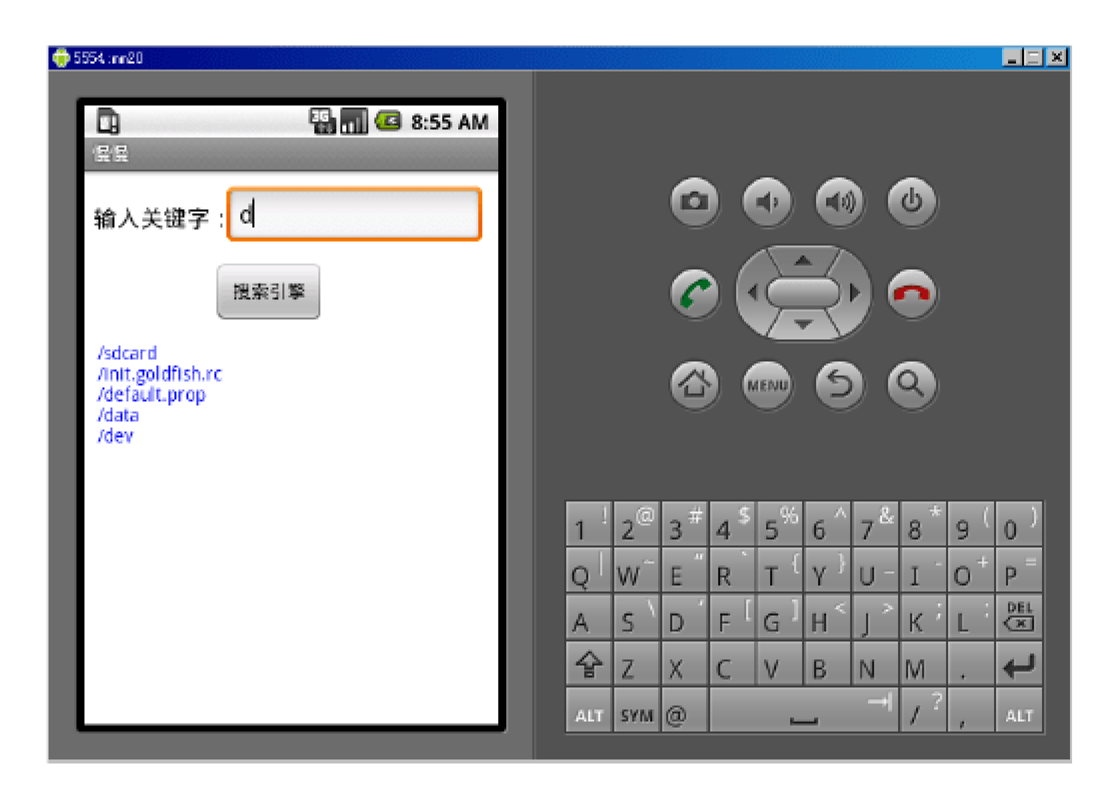


图 6-34　搜索的信息



第 7 章　数 据 存 储

在本书前面的章节中，已经多次涉及 Android 的数据存储知识。数据存储是手机领域中最常见的应用之一，通过数据存储能够在移动设备中显示不同的信息。在本章的内容中，将详细讲解 Android 数据存储的基本知识，并通过具体实例的实现过程来讲解其使用流程。

7.1　五种存储方式

Android 操作系统提供了一种公共文件系统，即任何应用软件都可以使用它来存储和读取文件，该文件也可以被其他的应用软件所读取(会有一些权限控制设定)。Android 采用了一种不同的系统，在 Android 中，所有的应用软件数据(包括文件)为该应用软件私有。然而，Android 同样也提供了一种标准方式供应用软件将私有数据开放给其他应用软件。在本章的内容中，将会描述一个应用软件存储和获取数据、开放数据给其他应用软件，以及从其他应用软件请求数据并且开放它们的多种方式。在 Android 中提供了如下 5 种存储方式：

(1) 文件存储；

(2) SQLite 数据库方式；

(3) 内容提供器(Content provider)；

(4) 网络；

(5) SharedPreferences。

7.2　SharedPreferences 是最简单的存储

在 Android 中，最简单的存储方式是 SharedPreferences，秘籍中对此方式的描述如下。

SharedPreferences 是 Android 提供用来存储一些简单配置信息的一种机制。例如，一些常见的欢迎语、登录用户名和密码等。SharedPreferences 是以键值对的方式存储，这样开发人员可以很方便地实现读取和存入。

7.2.1 SharedPreferences 存储类效率

SharedPreferences 是 Android 平台上的一个轻量级存储类，用于保存一些常用的配置信息，例如窗口状态。SharedPreferences 提供了 Android 平台常规的 Long(长整形)、Int(整形)、String(字符串形)的保存。SharedPreferences 类似 Windows 系统上的 ini 配置文件，但是它分为多种权限，可以全局共享访问，最终以 xml 方式来保存，但是整体效率不是特别高。XML 处理时，Dalvik 会通过自带底层的本地 XML Parser 解析，比如 XMLpull 方式，这样对于内存资源占用会比较好。

两个 Activity 之间的数据传递除了可以通过 intent 来传递外，还可以使用 SharedPreferences 来共享数据方式。SharedPreferences 用法很简单，例如在 A 中设置如下代码：

```
Editor sharedata = getSharedPreferences("data", 0).edit();
sharedata.putString("item","hello getSharedPreferences");
sharedata.commit();
```

然后即可在 B 中编写如下获取代码：

```
SharedPreferences sharedata = getSharedPreferences("data", 0);
String data = sharedata.getString("item", null);
Log.v("cola","data="+data);
```

最后通过如下 Java 代码将数据显示出来：

```
<SPAN class=hilite1>SharedPreferences</SPAN> sharedata = getSharedPreferences
("data", 0);
String data = sharedata.getString("item", null);
Log.v("cola","data="+data);
```

SharedPreferences 的用法基本上和 J2SE(java.util.prefs.Preferences)中的用法一样，以一种简单、透明的方式来保存一些用户个性化设置的字体、颜色、位置等参数信息。一般的应用程序都会提供“设置”或者“首选项”之类的界面，这样就可以通过 Preferences 来保存这些设置，而程序员不需要知道它到底以什么形式保存，保存在什么地方。

注意： 在 Android 系统中，这些信息以 XML 文件的形式保存在“/data/data/PACKAGE_NAME /shared_prefs”目录下。

7.2.2 演练

将 SharedPreferences 心法融会贯通之后，我决定开始演练一番。

题 目	目 的	源码路径
演练 1	实战演练 SharedPreferences 的使用过程	“光盘:\daima\7\UserSharedPreferences”文件夹

第 1 步：编写文件 SharedPreferencesHelper.java，主要代码如下所示：

```
public class SharedPreferencesHelper {
    SharedPreferences sp;
```

```
    SharedPreferences.Editor editor;
        Context context;
        public SharedPreferencesHelper(Context c,String name){
        context = c;
        sp = context.getSharedPreferences(name, 0);
        editor = sp.edit();
    }
        public void putValue(String key, String value){
        editor = sp.edit();
        editor.putString(key, value);
        editor.commit();
    }
    public String getValue(String key){
        return sp.getString(key, null);
    }
    }
```

第 2 步：编写文件 SharedPreferencesUsage.java，主要代码如下所示：

```
    public class SharedPreferencesUsage extends Activity {
    public final static String COLUMN_NAME ="name";
    public final static String COLUMN_MOBILE ="mobile";

    SharedPreferencesHelper sp;
    /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      //setContentView(R.layout.main);
      sp = new SharedPreferencesHelper(this, "contacts");
      //1. to store some value
      sp.putValue(COLUMN_NAME, "张三");
      sp.putValue(COLUMN_MOBILE, "00000000000");

      String name = sp.getValue(COLUMN_NAME);
      String mobile = sp.getValue(COLUMN_MOBILE);

      TextView tv = new TextView(this);
      tv.setText("NAME:"+ name + "\n" + "MOBILE:" + mobile);
            setContentView(tv);
   }
}
```

至此，整个演练就结束了，执行后的效果如图 7-1 所示。这样 “NAME”和“MOBILE”就存储在 SharedPreferences 中了。

因为上面实例中的 pack_name 为 package com.android.SharedPreferences，所以存放数据的路径为：data/data/com.android.SharedPreferences/share_prefs/contacts.xml，文件 contacts.xml 的内容如下所示。

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
```

```
<string name="mobile">123456789</string>
<string name="name">Gryphone</string>
</map>
```

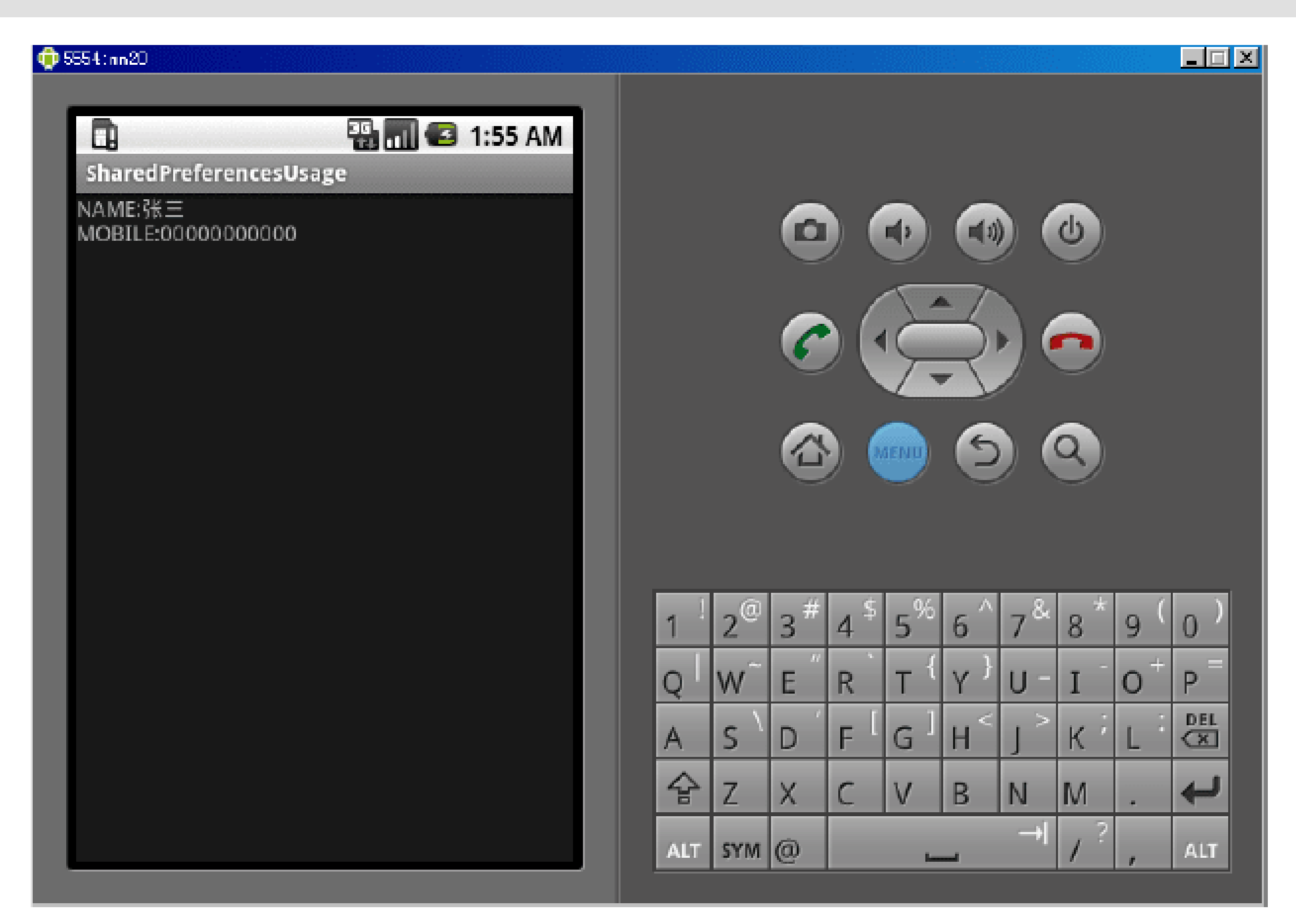


图 7-1　执行效果

7.3　最危险的地方最安全

前面介绍的 Shared Preferences 存储方式非常方便，但是只适用于存储比较简单的数据，如果需要存储更多的数据，Android 中可供选择的方式有好几种，这里先给读者介绍文件存储的方法。同传统的 Java 中实现 I/O 的程序类似，在 Android 中提供了 openFileInput 和 openFileOuput 方法来读取设备上的文件，例如如下所示的代码：

```
String FILE_NAME = "tempfile.tmp";                    //确定要操作文件的文件名
  //初始化
  FileOutputStream fos = openFileOutput(FILE_NAME, Context.MODE_PRIVATE);
  FileInputStream fis = openFileInput(FILE_NAME);       //创建写入流代码解释:
```

在上述代码中，通过这两个方法只支持读取该应用目录下的文件，读取非自身目录下的文件将会抛出异常。需要提醒的是：如果调用 FileOutputStream 时指定的文件不存在，Android 会自动创建它。另外，在默认情况下，写入的时候会覆盖原文件内容，如果想把新写入的内容附加到原文件内容后，则可以指定其模式为 Context.MODE_APPEND。在默认情况下，使用 OpenFileOutput 方法创建的文件只能被其调用的应用使用，其他应用无法读取这个文件，如果需要在不同的应用中共享数据，可以使用 Content Provider 实现。

如果应用需要一些额外的资源文件，例如，一些用来测试音乐播放器是否可以正常工作的 MP3 文件，可以将这些文件放在应用程序的/res/raw/目录下，例如，mydatafile.mp3。那么就可以在应用中使用 getResources 方法获取资源后，用 openRawResource 方法(不带后缀的资源文件名)打开这个文件，实现代码如下所示：

```
Resources myResources = getResources();
   InputStream myFile = myResources.openRawResource(R.raw.myfilename);
```

除了前面介绍的读写文件外，Android 中还提供了诸如 deleteFile、fileList 等方法来操作文件。

7.4　藏经阁和 SQLite

藏经阁，又称法堂，是寺院讲经说法藏经的场所。秘籍中说道藏经阁是少林的武学宝库，寺中的大多数武学典籍会被藏于此。如果将藏经阁和 SQLite 做一个简单对比：藏经阁秘籍奇多，是一个重量级的存储方式；而安卓中最通用的存储方式是 SQLite 存储，SQLite 是 Android 自带的一个标准的数据库，支持 SQL 语句，它是一个轻量级的嵌入式数据库。

题　目	目　的	源码路径
演练 2	实战演练 SQLite 的使用流程	“光盘:\daima\7\UserSQLite”文件夹

编写的主文件是 UserSQLite，具体实现流程如下。

(1)　DatabaseHelper 类继承 SQLiteOpenHelper，具体代码如下所示：

```
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE " + TABLE_NAME + " (" + TITLE
                + " text not null, " + BODY + " text not null " + ");";
        Log.i("haiyang:createDB=", sql);
        db.execSQL(sql);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

在上述代码中，DatabaseHelper 类继承了 SQLiteOpenHelper 类，并且重写了 onCreate 和 onUpgrade 方法。在 onCreate()方法里首先我们构造了一条 SQL 语句，然后调用 db.execSQL(sql)执行 SQL 语句。这条 SQL 语句为我们生成了一张数据库表。

上述代码实现了两个函数，各个函数的具体说明如下。

- onCreate(SQLiteDatabase)：在数据库第一次生成的时候会调用这个方法，一般在这个方法中生成数据库表。
- onUpgrade(SQLiteDatabase, int, int)：当数据库需要升级的时候，Android 系统会主动地调用这个方法。一般在这个方法中删除数据表，并建立新的数据表，当然是否还需要做其他操作，完全取决于应用的需求。

(2)　编写按钮处理事件，单击“插入两条数据”按钮，如果数据成功插入到数据库中的 diary 表中，那么在界面的 title 区域就会有成功的提示，如图 7-2 所示。



图 7-2　插入成功

当单击“插入两条数据”按钮后，会执行监听器里的 onClick 方法，并最终执行了程序中的 insertItem 方法。其具体代码如下所示：

```
    /*
    * 插入两条数据
    */
   private void insertItem() {
         /得到一个可写的 SQLite 数据库，如果这个数据库还没有建立/
   /那么 mOpenHelper 辅助类负责建立这个数据库。/
   /如果数据库已经建立，那么直接返回一个可写的数据库。/
       SQLiteDatabase db = mOpenHelper.getWritableDatabase();
       String sql1 = "insert into " + TABLE_NAME + " (" + TITLE + ", " + BODY
               + ") values('AA', 'android好好好好好好好');";
       String sql2 = "insert into " + TABLE_NAME + " (" + TITLE + ", " + BODY
               + ") values('BB', 'android好好好好好好好');";
       try {
           Log.i("haiyang:sql1=", sql1);
           Log.i("haiyang:sql2=", sql2);
           db.execSQL(sql1);
           db.execSQL(sql2);
           setTitle("插入成功");
       } catch (SQLException e) {
           setTitle("插入失败");
       }
   }
```

注意： Android 支持 5 种打印输出级别，分别是 Verbose、Debug、Info、Warning、Error，当然我们在程序中一般用到的是 Info 级别，即将一些自己需要知道的信息打印出来，如图 7-3 所示。

```
Problems  Javadoc  Declaration  Properties  Console
Android
[2010-02-19 15:50:15 - SQLite] Uploading SQLite.apk onto device 'emulator-5554'
[2010-02-19 15:50:15 - SQLite] Installing SQLite.apk...
[2010-02-19 15:50:31 - SQLite] Success!
[2010-02-19 15:50:32 - SQLite] Starting activity com.eoeAndroid.SQLite.ActivityMain on device
[2010-02-19 15:50:39 - SQLite] ActivityManager: Starting: Intent { comp={com.eoeAndroid.SQLite/com.eoeAndroid.SQLite.Ac
```

图 7-3　打印输出级别

(3) 单击“查询数据库”按钮，会在界面的 title 区域显示当前数据表中数据的条数，刚才我们插入了两条，那么现在单击后应该显示为两条，如图 7-4 所示。

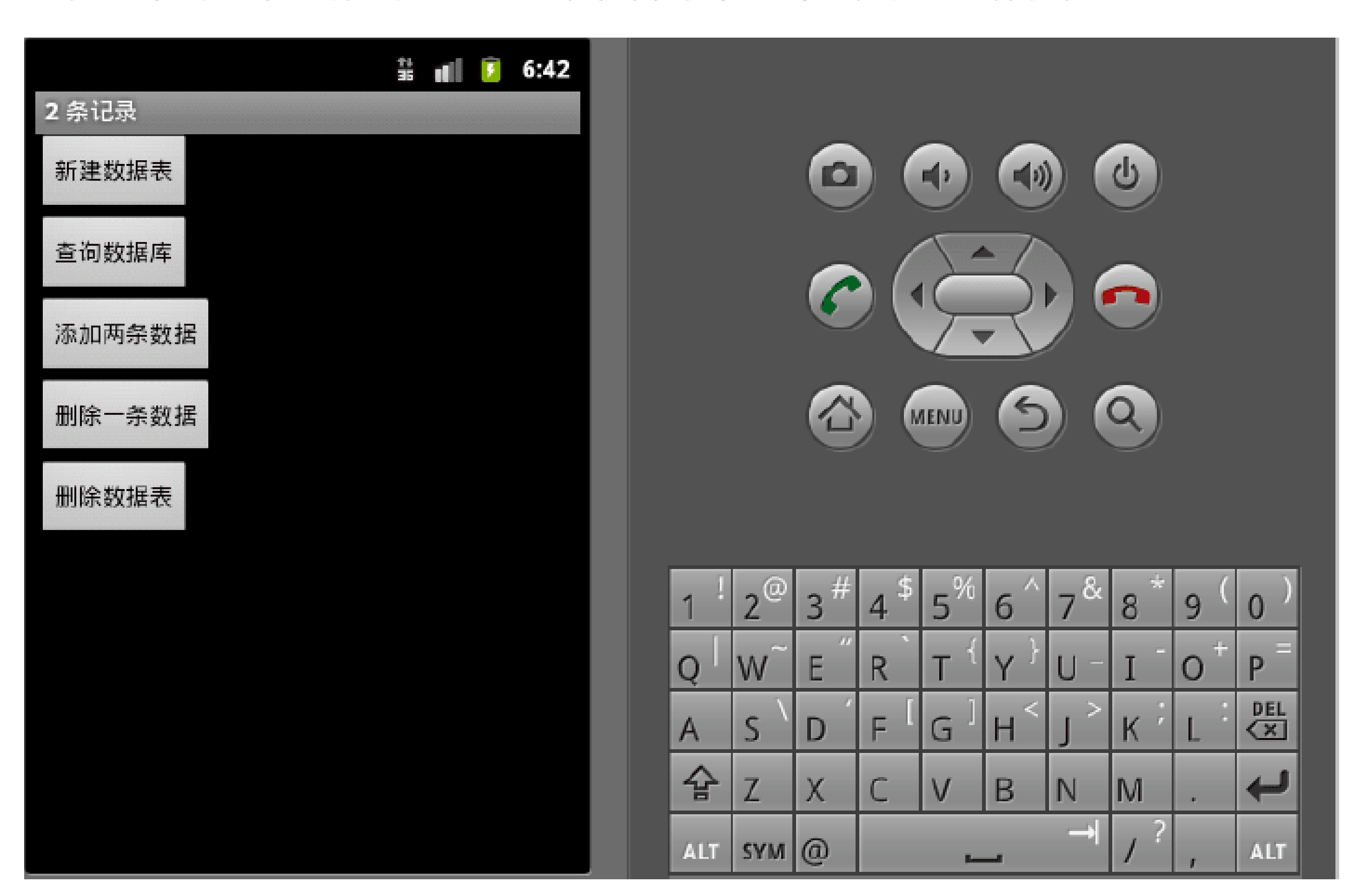


图 7-4　查询数据

单击“查询数据库”按钮后，程序执行了监听器里的 onClick 方法，并最终执行了上述程序里的 showItems 方法，具体代码如下所示：

```
     /*
     * 在屏幕的 title 区域显示当前数据表当中的数据条数。
     */
    private void showItems() {
        /得到一个可写的数据库/
        SQLiteDatabase db = mOpenHelper.getReadableDatabase();
        String col[] = { TITLE, BODY };
        Cursor cur = db.query(TABLE_NAME, col, null, null, null, null, null);
        /通过 getCount()方法，可以得到 Cursor 当中数据的个数。/
        Integer num = cur.getCount();
        setTitle(Integer.toString(num) + " 条记录");
    }
}
```

在上述代码中，语句“Cursor cur = db.query(TABLE_NAME, col, null, null, null, null, null)”比较难以理解，此语句用于把查询到的数据放到一个 Cursor 当中。Cursor 里封装了这个数据表 TABLE_NAME 中的所有条例。在 Android 中 query()方法相当有用，它包含了 7 个参数，各个参数的具体说明如下。

- ❑ 第 1 个参数是数据库里表的名字，比如在这个例子中，表的名字就是 TABLE_NAME，也就是"diary"。
- ❑ 第 2 个字段是想要返回数据包含的列的信息。在这个例子中想要得到的列有 title、body，我们把这两个列的名字放到字符串数组中。
- ❑ 第 3 个参数为 selection，相当于 SQL 语句的 where 部分，如果想返回所有的数据，那么就直接置为 null。
- ❑ 第 4 个参数为 selectionArgs，在 selection 部分，你有可能用到"?"，那么在 selectionArgs 定义的字符串会代替 selection 中的"?"。

- ❑ 第 5 个参数为 groupBy，定义查询出来的数据是否分组，如果为 null 则说明不用分组。
- ❑ 第 6 个参数为 having ， 相当于 SQL 语句中的 having 部分。
- ❑ 第 7 个参数为 orderBy，来描述期望返回值是否需要排序，如果设置为 null 则说明不需要排序。

注意：Cursor 在 Android 中是一个非常有用的接口，通过 Cursor 我们可以对从数据库查询出来的结果集进行随机的读写访问。

(4) 单击“删除一条数据”按钮后，如果成功删除，我们可以看到在屏幕的标题(title)区域有文字提示，如图 7-5 所示。如果再次单击“查询数据库”按钮，查询数据库中的记录是不是少了一条。单击“查询数据库”按钮后，出现如图 7-6 所示的界面。



图 7-5　删除一条数据



图 7-6　查询数据库

当单击“删除一条数据”按钮后，程序执行监听器里的 onClick 方法，并最终执行了上述程序中的 deleteItem 方法。其代码如下所示：

```
    /*
    * 删除其中的一条数据
    */
   private void deleteItem() {
       try {
           SQLiteDatabase db = mOpenHelper.getWritableDatabase();
           db.delete(TABLE_NAME, " title = 'AA'", null);
           setTitle("删除 title 为 AA 的一条记录");
       } catch (SQLException e) {
       }
   }
```

在上述代码中，通过“db.delete(TABLE_NAME, " title = 'haiyang'", null)”语句删除一条 title='haiyang'的数据。当然如果有很多条数据 title 都为'haiyang'，那么一并删除。delete 方法各个参数的具体说明如下。

- ❑ 第一个参数是数据库表名，在这里是 TABLE_NAME，也就是 diary。
- ❑ 第二个参数相当于 SQL 语句中的 where 部分，也就是描述了删除的条件。

如果在第二个参数中有"？"符号，那么第三个参数中的字符串会依次替换在第二个参数中

出现的"？"符号。

(5)　单击“删除数据表”按钮，可以删除 diary 这张数据表，如图 7-7 所示。

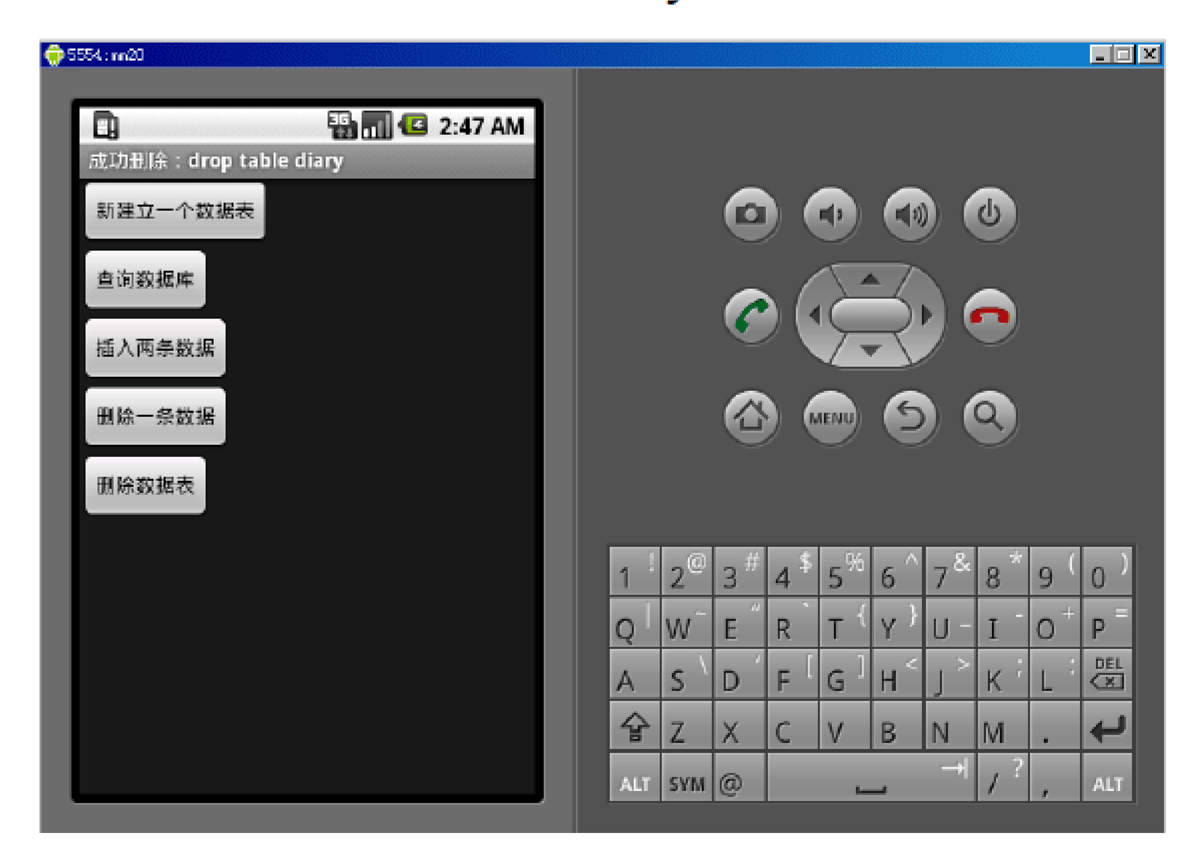


图 7-7　删除数据表

数据库表是怎么实现删除的，具体代码如下所示：

```
    /*
    * 删除数据表
    */
   private void dropTable() {
       SQLiteDatabase db = mOpenHelper.getWritableDatabase();
       String sql = "drop table " + TABLE_NAME;
       try {
           db.execSQL(sql);
           setTitle("成功删除: " + sql);
       } catch (SQLException e) {
           setTitle("删除错误");
       }
   }
```

在上述代码中，构造了一个标准的删除数据表的 SQL 语句，然后执行这条语句 db.execSQL(sql)。

(6)　单击其他按钮，程序运行时有可能会出现异常情况，在此单击“新建立一个数据表”按钮，如图 7-8 所示。单击“查询数据库”按钮，查询数据库中是否有数据，如图 7-9 所示。



图 7-8　新建数据表



图 7-9　显示 0 条记录

如何建立一张新数据表的，具体的实现代码如下所示：

```
    * /*
    * 重新建立数据表
    */
   private void CreateTable() {
       SQLiteDatabase db = mOpenHelper.getWritableDatabase();
       String sql = "CREATE TABLE " + TABLE_NAME + " (" + TITLE
               + " text not null, " + BODY + " text not null " + ");";
       Log.i("haiyang:createDB=", sql);
       try {
           db.execSQL("DROP TABLE IF EXISTS diary");
           db.execSQL(sql);
           setTitle("数据表成功重建");
       } catch (SQLException e) {
           setTitle("数据表重建错误");
       }
   }
```

在上述代码中，SQL 变量表示的语句为标准的 SQL 语句，负责按要求建立一张新数据表。“db.execSQL("DROP TABLE IF EXISTS diary")”语句表示：如果存在 diary 表，则需要先删除它，因为在同一个数据库中不能出现两张同样名字的表；“db.execSQL(sql)”语句用于执行 SQL 语句，建立一张新数据库表。

7.5 峰 回 路 转

在 Android 系统中，数据是私有的，当然这些数据包括文件数据和数据库数据以及一些其他类型的数据。Android 中的两个程序之间可以进行数据交换，此功能就是通过 ContentProvider 实现的。在本节的内容中，简要介绍 Content Provider 存储的基本知识。

7.5.1 ContentProvider

一个 Content Provider 类实现了一组标准的方法接口，从而能够让其他的应用保存或读取此 Content Provider 的各种数据类型。也就是说，一个程序可以通过实现一个 Content Provider 的抽象接口将自己的数据暴露出去。外界根本看不到，也不用看到这个应用暴露的数据在应用中是如何存储的，或者是用数据库存储还是用文件存储，或者是通过网上获得，这一切都不重要，重要的是外界可以通过这一套标准及统一的接口和程序里的数据打交道，可以读取程序的数据，也可以删除程序的数据，当然，中间也会涉及一些权限的问题。现实中比较常见的接口如下。

(1) ContentResolver 接口

外界的程序通过 ContentResolver 接口可以访问 ContentProvider 提供的数据，在 Activity 当中通过 getContentResolver()可以得到当前应用的 ContentResolver 实例。ContentResolver 提供的接口和 ContentProvider 中需要实现的接口对应，具体来说主要有以下几个。

- query(Uri uri, String[] projection, String selection, String[] selectionArgs,String sortOrder)：通过 Uri 进行查询，返回一个 Cursor。
- insert(Uri uri, ContentValues values)：将一组数据插入到 Uri 指定的地方。

- ❑ update(Uri uri, ContentValues values, String where, String[] selectionArgs)：更新 Uri 指定位置的数据。
- ❑ delete(Uri uri, String where, String[] selectionArgs)：删除指定 Uri 并且符合一定条件的数据。

(2) ContentProvider 和 ContentResolver 中的 Uri

在 ContentProvider 和 ContentResolver 中，使用的 Uri 的形式通常有两种，一种是指定全部数据；另一种是指定某个 ID 的数据。具体见下面的实例。

```
content://contacts/people/      //此 Uri 指定的就是全部的联系人数据
content://contacts/people/1     //此 Uri 指定的是 ID 为 1 的联系人数据
```

在上边两个类中用到的 Uri 一般由如下 3 个部分组成。

- ❑ 第一部分是："content://"。
- ❑ 第二部分是要获得数据的一个字符串片段。
- ❑ 第三部分是 ID(如果没有指定 ID，那么表示返回全部)。

因为 URI 通常比较长，而且有时候容易出错且难以理解。所以，在 Android 当中定义了一些辅助类，并且定义了一些常量来代替这些长字符串的使用，例如如下的代码：

```
Contacts.People.CONTENT_URI (联系人的 URI)
```

7.5.2　实战演练 ContentProvider

为了将前面所学的知识达到融会贯通的程序，在接下来的内容中，将通过一个具体实例来讲解使用 ContentProvider 进行存储操作的基本方法。

题　目	目　的	源码路径
演练 3	实战演练 ContentProvider 的使用流程	"光盘:\daima\7\UseContentProvider" 文件夹

编写主程序 ActivityMain 中的 onCreate()方法，其具体代码如下所示：

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Cursor c = getContentResolver().query(Phones.CONTENT_URI, null, null,
null, null);
        startManagingCursor(c);
        ListAdapter adapter = new SimpleCursorAdapter(this,
                android.R.layout.simple_list_item_2, c,
                        new String[] { Phones.NAME, Phones.NUMBER },
                        new int[] { android.R.id.text1, android.R.id.text2 });
        setListAdapter(adapter);
    }
```

关于上述代码的解释如下。

(1) getContentResolver()方法：得到应用的 ContentResolver 实例。

(2) query(Phones.CONTENT_URI, null, null, null, null)：是 ContentResolver 中的方法，负责查询所有联系人，并返回一个 Cursor。这个方法参数比较多，各个参数的具体含义如下。

- ❑ 第 1 个参数为 Uri，在这个例子里 Uri 是联系人的 Uri。

- ❑ 第 2 个参数是一个字符串的数组，数组里边的每一个字符串都是数据表中某一列的名字，它指定返回数据表中那些列的值。
- ❑ 第 3 个参数相当于 SQL 语句的 where 部分，描述哪些值是我们需要的。
- ❑ 第 4 个参数是一个字符串数组，它里边的值依次代替在第三个参数中出现的“?”符号。
- ❑ 第 5 个参数指定了排序的方式。

(3) startManagingCursor(c)语句：让系统来管理生成的 Cursor。

(4) ListAdapter adapter = new SimpleCursorAdapter(this,Android.R.layout.simple_list_item_2, c, new String[] { Phones.NAME, Phones.NUMBER }, new int[] { Android.R.id.text1, Android.R.id.text2 })语句：用于生成一个 SimpleCursorAdapter。

(5) setListAdapter(adapter)：将 ListView 和 SimpleCursorAdapter 进行绑定。

运行后将会看到如图 7-10 所示的界面。

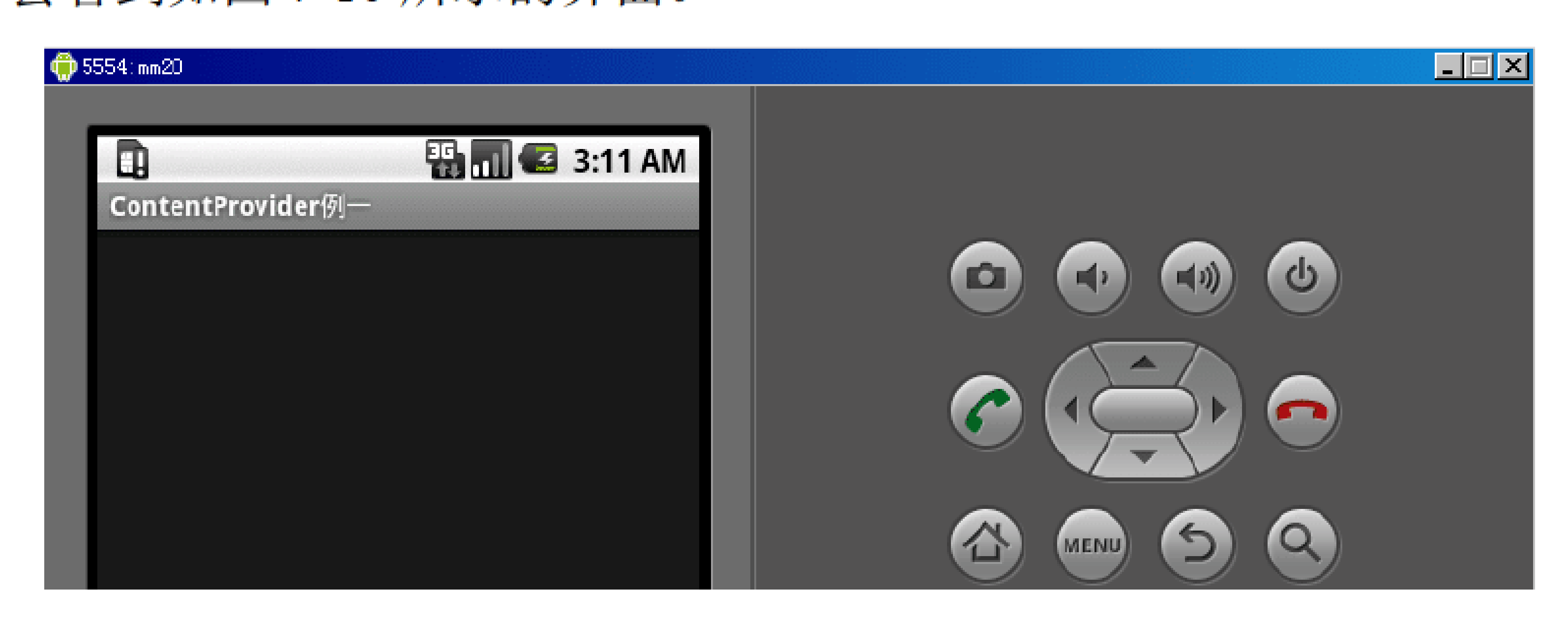


图 7-10 运行效果

可以添加几条数据到联系人列表中，具体流程如下。

(1) 单击模拟器的键，在转出来的界面上，单击桌面上的 Contacts 图标，如图 7-11 所示。

(2) 进入应用程序后，单击 MENU 按钮，在出现的界面上单击 Create new contact 按钮，如图 7-12 所示。

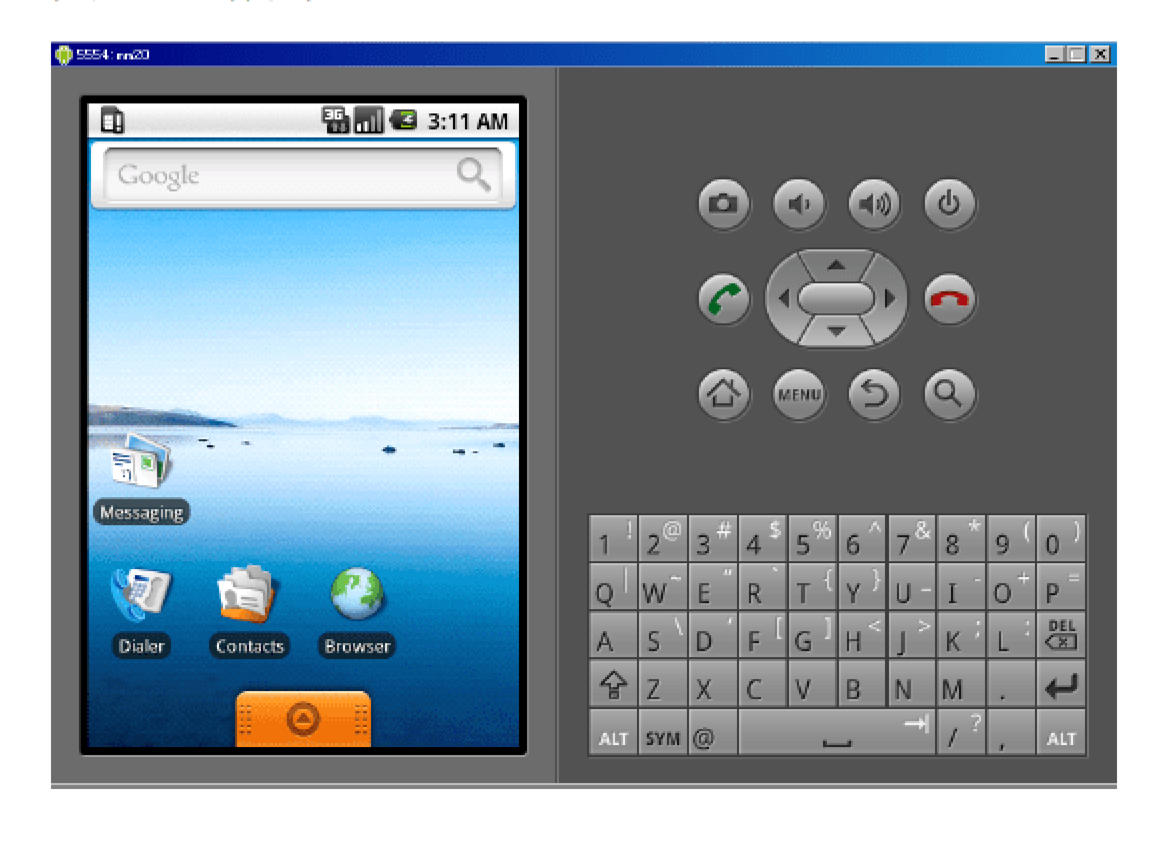


图 7-11 出现的桌面

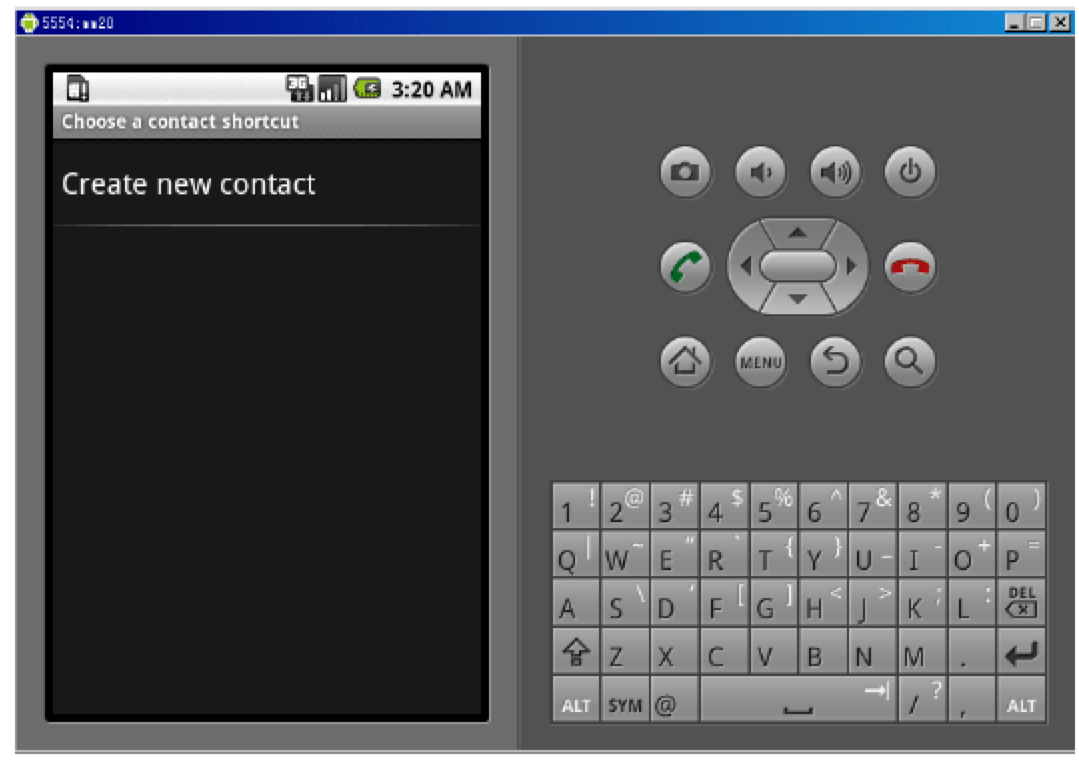


图 7-13 单击 Create new contact 按钮

(3) 添加联系人的姓名和电话号码信息，如图 7-13 所示。

(4) 单击 MENU 按钮，在返回的界面上单击 Save 按钮保存，如图 7-14 所示。

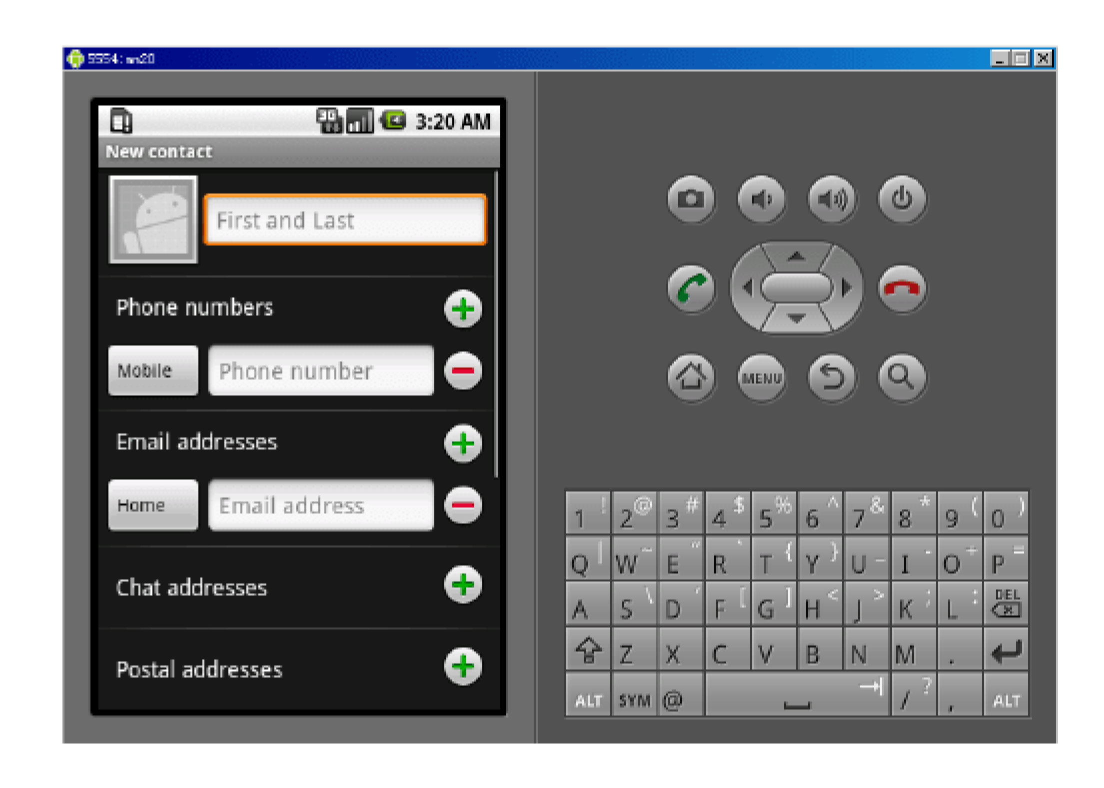


图 7-13　添加联系人姓名和电话号码

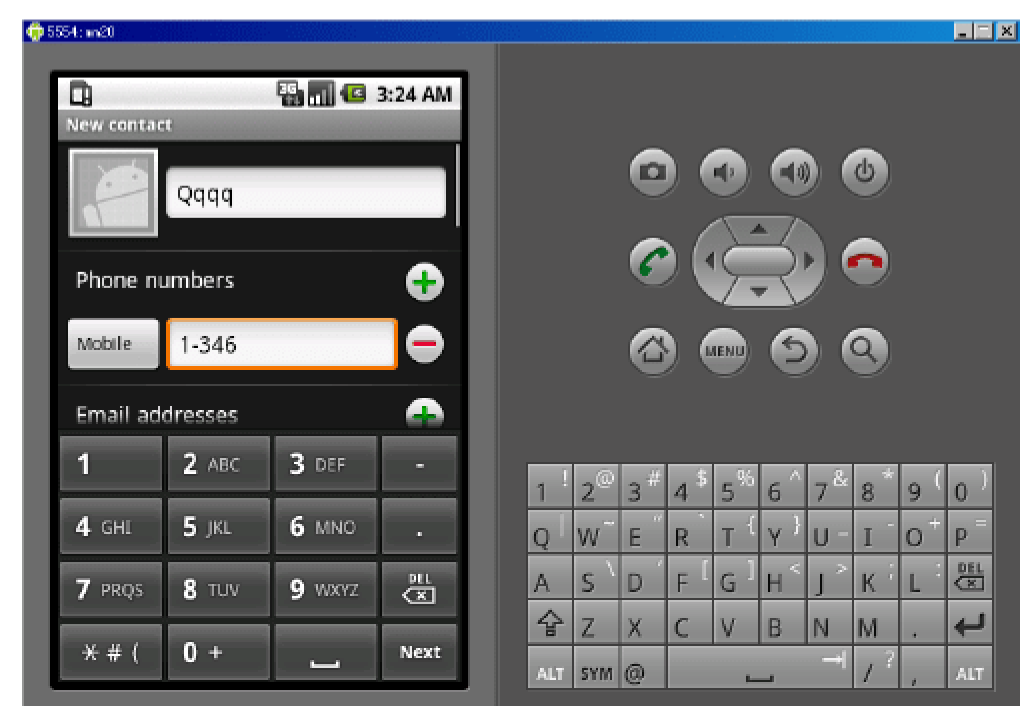


图 7-14　单击 Save 按钮保存

(5)　按照上述操作步骤，即可添加一条联系人数据，效果如图 7-15 所示。

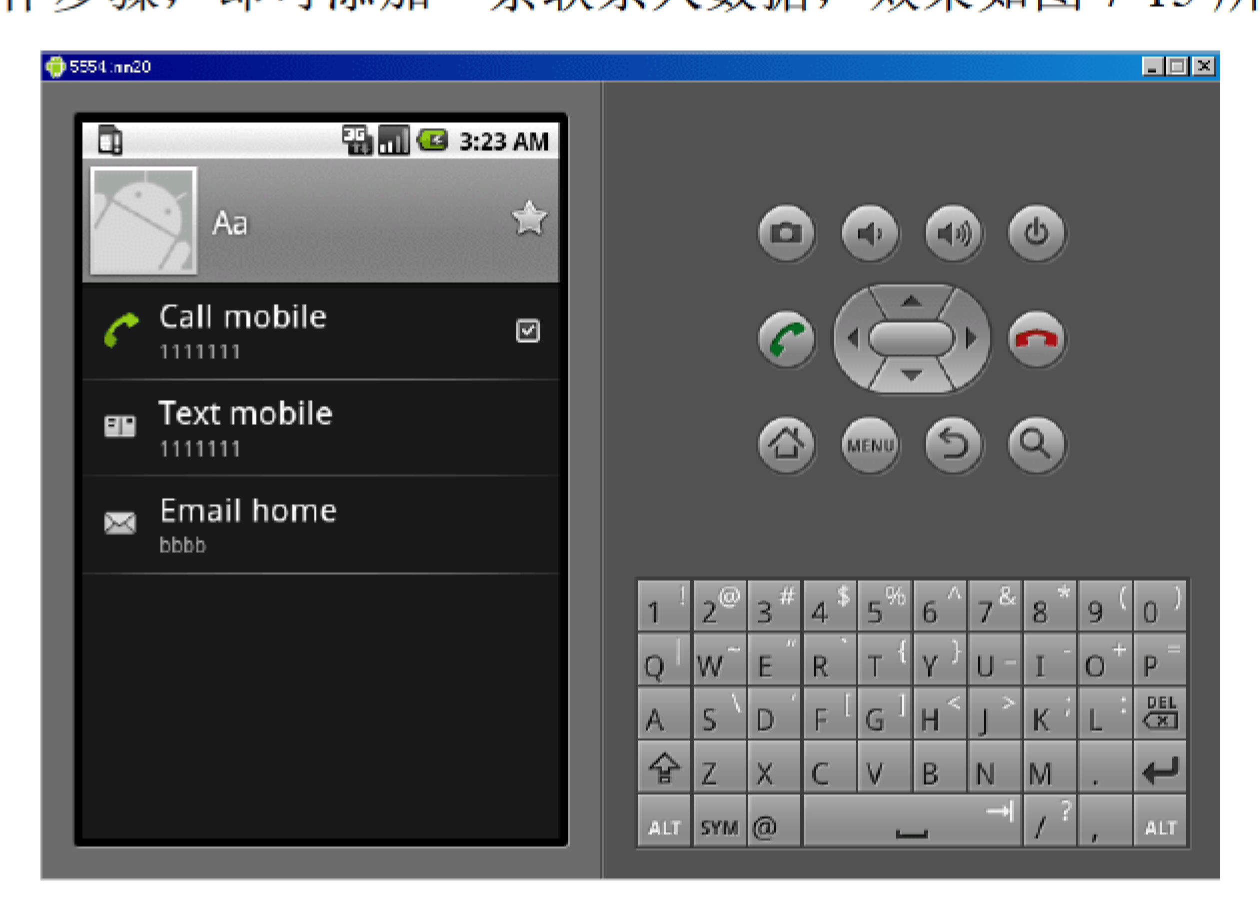


图 7-15　添加数据后的效果

7.6　网 络 存 储

在安卓中还有一种存储(获取)数据的方式，即通过网络来实现数据的存储和获取。在 Android 的早期版本中，曾经支持过进行 XMPP Service 和 Web Service 的远程访问。Android SDK 1.0 以后的版本对它以前的 API 作了许多变更。Android 1.0 以上版本不再支持 XMPP Service，访问 Web Service 的 API 也全部进行了变更。

1．演练简介

本实例的功能是通过邮政编码查询该地区的天气预报，以 POST 的方式发送请求到 webservicex.net 站点，访问 WebService.webservicex.net 站点上提供的查询天气预报服务，具体信息请参考 WSDL 文档，网址如下：

```
http://www.webservicex.net/WeatherForecast.asmx?WSDL
```

输入：美国某个城市的邮政编码。

输出：该邮政编码对应城市的天气预报。

2. 实现过程

具体实现过程如下。

(1) 如果需要访问外部网络，则需要在 AndroidManifest.xml 文件中加入如下代码申请权限许可：

```
<!-- Permissions -->
  <uses-permission Android:name="Android.permission.INTERNET" />
```

(2) 以 HTTP POST 的方式发送，SERVER_URL 并不是指 WSDL 的 URL，而是服务本身的 URL。具体实现的代码如下所示：

```
private  static  final  String  SERVER URL  =  "http://www.webservicex.net/
WeatherForecast. asmx/GetWeatherByZipCode";    //定义需要获取的内容来源地址
  HttpPost request = new HttpPost(SERVER_URL); //根据内容来源地址创建一个 Http 请求
  // 添加一个变量
  List <NameValuePair> params = new ArrayList <NameValuePair>();
  // 设置一个华盛顿区号
  params.add(new BasicNameValuePair("ZipCode", "200120"));  //添加必需的参数
  request.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF_8)); //设置参数的
                                                                    编码
  try {  HttpResponse httpResponse = new DefaultHttpClient().execute(request);
//发送请求并获取反馈
  // 解析返回的内容
  if(httpResponse.getStatusLine().getStatusCode() != 404)
  {
    String result = EntityUtils.toString(httpResponse.getEntity());
    Log.d(LOG_TAG, result);
  }
  } catch (Exception e) {
  Log.e(LOG_TAG, e.getMessage());
  }
```

通过上述代码，使用 Http 从 webservicex 获取 ZipCode 为"200120"(美国 WASHINGTON DC)的内容，其返回的内容如下所示：

```
<WeatherForecasts   xmlns:xsd="http://www.w3.org/2001/XMLSchema"   xmlns:xsi=
"http: //www.w3.org/2001/XMLSchema-instance" xmlns="http://www.webservicex.net">
    <Latitude>38.97571</Latitude>
    <Longitude>77.02825</Longitude>
    <AllocationFactor>0.024849</AllocationFactor>
    <FipsCode>11</FipsCode>
    <PlaceName>WASHINGTON</PlaceName>
    <StateCode>DC</StateCode>
    <Details>
      <WeatherData>
        <Day>Saturday, April 25, 2009</Day>
       <WeatherImage>http://forecast.weather.gov/images/wtf/sct.jpg</WeatherImage>
        <MaxTemperatureF>88</MaxTemperatureF>
        <MinTemperatureF>57</MinTemperatureF>
        <MaxTemperatureC>31</MaxTemperatureC>
        <MinTemperatureC>14</MinTemperatureC>
```

```
    </WeatherData>
    <WeatherData>
      <Day>Sunday, April 26, 2009</Day>

<WeatherImage>http://forecast.weather.gov/images/wtf/few.jpg</WeatherImage>
      <MaxTemperatureF>89</MaxTemperatureF>
      <MinTemperatureF>60</MinTemperatureF>
      <MaxTemperatureC>32</MaxTemperatureC>
      <MinTemperatureC>16</MinTemperatureC>
    </WeatherData>
  …
   </Details>
  </WeatherForecasts>
```

通过上述实例，演示了如何在 Android 中通过网络获取数据。要掌握该类内容，开发者需要熟悉 java.net.*，Android.net.*这两个包中的内容，具体请读者参阅相关文档。



第 8 章　电话与短信双剑合璧

在本章的内容中，将进一步介绍 Intent 的使用方法，通过电话和短信两个实例讲解 Android 中基本应用的开发。打电话和发短信是任何一款智能手机的基本功能，它需要手机平台底层(GSM/3G 模块)的支持，因此本章就通过电话和短信的例子来展示应用程序该如何利用 Android 提供的 API 同设备上的通信模块打交道。

8.1　电话和短信天生是一对

在安卓的众多应用中有一对侠侣——短信和电话，它们是天生的一对。这两种功能是基于 Intent 实现的，Intent 是一种运行时的绑定(runtime binding)机制，它能在程序运行的过程中连接两个不同的组件。通过 Intent，程序可以向 Android 表达某种请求或者意愿，Android 会根据意愿的内容选择适当的组件来完成请求。比如，有一个 Activity 希望打开网页浏览器查看某一网页的内容，那么这个 Activity 只需要发出 WEB_SEARCH_ACTION 请求给 Android，Android 会根据 Intent 的请求内容，查询各组件注册时声明的 IntentFilter，找到网页浏览器的 Activity 来浏览网页。

8.1.1　怀念昨日之 Intent

都知道 Android 中有 3 个基本组件——Activity、Service 和 BroadcastReceiver，都是通过 Intent 机制激活的，而不同类型的组件有传递 Intent 的不同方式，具体过程说明如下。

(1)　要激活一个新的 Activity，或者让一个现有的 Activity 执行新的操作，可以通过调用 Context.startActivity()或 Activity.startActivityForResult()方法。这两个方法需要传入的 Intent 参数也称为 Activity Action Intent(活动行为意图)，根据 Intent 对象对目标 Activity 描述的不同，来启动与之相匹配的 Activity 或传递信息。

(2)　要启动一个新的服务，或者向一个已有的服务传递新的指令，调用 Context.startService()方法或 Context.bindService()方法可以将调用此方法的上下文对象与 Service 绑定。

(3)　分别通过 Context.sendBroadcast()、Context.sendOrderBroadcast()和 Context.sendStickBroadcast()

三个方法，可以发送 BroadcastIntent。当 BroadcastIntent 发送后，所有已注册的拥有与 IntentFilter 相匹配的 BroadcastReceiver 就会被激活，这种机制被广泛地应用于设备或系统状态变化的通知。例如，当 Android 的电池电量过低时，系统会发送 Action 为 BATTERY_LOW 的广播，接着任何可匹配该 Action 的 IntentFilter 注册的 BroadcastReceiver 都会各自运行自定义的处理代码，比如关闭设备的 WIFI 和 GPS 以节省电池消耗。

当 Intent 发出后，Android 会准确找到相匹配的一个或多个 Activity、Service 或 BroadcastReceiver 作为响应。所以，不同类型的 Intent 消息不会出现重叠，而不可能发送给 Activity 或 Service。因为 startActivity()传递的消息也只可能发送给 Activity，由 startService()传递的 Intent 只可能发送给 Service。

8.1.2 Intent 组成知多少

Intent 对象抽象地描述了要执行的操作，其描述的基本内容可以分为组件名称、Action(动作)、Data(数据)、Category(类别)、Extra(附加信息)和 Flag(标志位)6 部分，下面将详细介绍。

(1) 组件名称是指 Intent 目标组件的名称。组件名称是一个 ComponentName 对象，这种对象名称是目标组件类名和目标组件所在应用程序的包名的组合。组件中包名不一定要和 manifest 文件中的包名完全匹配。组件名称是一个可选项，如果 Intent 消息中指明了目标组件的名称，这就是一个显式消息，Intent 会传递给指明的组件。如果目标组件名称没有指定，Android 则通过 Intent 内的其他信息和已注册的 IntentFilter 比较来选择合适的目标组件。

(2) Action 描述 Intent 所触发动作名字的字符串，对于 BroadcastIntent 来说，Action 指被广播出去的动作。理论上 Action 可以为任何字符串，而与 Android 系统应用有关的 Action 字符串以静态字符串常量的形式定义在 Intent 类中。Android 系统中常见的 Activity Action Intent 的 Action 如下所示。

- ❑ ACTION_ANSWER：打开一个 Activity 处理来电。目前，它是被本地的电话拨号工具处理。
- ❑ ACTION_CALL：启动电话拨号工具，并立即用数据 URI 中的号码初始化一个呼叫。一般来说，如果可能的话，它认为是比使用 Dial_Action 更好的一种方式。
- ❑ ACTION_DELETE：启动一个 Activity 来删除储存在 URI 位置的数据入口。
- ❑ ACTION_DIAL：启动一个电话拨号程序，使用预置在数据 URI 中的号码来拨号。默认情况下，它是由 Android 本地的电话拨号工具处理。这个拨号工具能规范多数的号码。例如，tel：555-1234 和 tel：(212)555 1212 都是有效的号码。
- ❑ ACTION_EDIT：请求一个 Activity 来编辑 URI 处的数据。
- ❑ ACTION_INSERT：打开一个能在数据域的特定游标处插入新项目的 Activity。当以子 Activity 方式调用时，它必须返回新插入项目的 URI。
- ❑ ACTION_PICK：启动一个子 Activity 从 URI 数据处挑选一个项目。当关闭时，它必须返回指向被挑选项目的 URI。启动的 Activity 取决于要挑选的数据，例如，传入 content://contacts/people 会引发本地的联系人列表。
- ❑ ACTION_SEARCH：启动一个 UI 来执行搜索，在 Intent 的数据包里使用

SearchManager.QUERY 键值来提供搜索内容的字符串。

- ACTION_SENDTO：启动一个 Activity 来给 URI 中的指定联系人发送一个消息。
- ACTION_SEND：启动一个 Activity 来发送特定的数据(接收者经由解析 Activity 来选择)。使用 setType 来设置 Intent 的类型为传输数据的 mime 类型。

数据本身依赖于类型使用 EXTRA_TEXT 或 EXTRA_STREAM 来储存。在 E-mail 的情况下，Android 本地应用程序还可以接受使用 EXTRA_EMAIL、EXTRA_CC、EXTRA_BCC 和 EXTRA_SUBJECT 键值的 EXTRAS。

- ACTION_VIEW：最通用的动作。View 动作要求 Intent URI 中的数据以最合理的方式显示。不同的应用程序将处理 View 请求，依赖于 URI 中的数据。通常，http：地址会在浏览器中打开；tel：地址会在拨号工具中打开并呼叫号码；geo：地址会在 Google 地图应用程序中显示，联系人内容会在联系人管理器中显示。
- ACTION_WEB_SEARCH：打开一个 Activity，执行基于数据 URI 中文本的网页搜索。和这些 Activity 动作一样，Android 本地应用程序还包括大量的 Broadcast 动作，用来创建 Intent 将系统消息通知给应用程序。这些 Broadcast 动作将在本章稍后部分进行讲解。常见的 BroadcastIntent Action 常量如下所示。
  - ACTION_TIME_TICK：当前时间改变，每分钟都发送，不能通过组件声明来接收，只有通过 Context.registerReceiver()方法来注册。
  - ACTION_TIME_CHANGED：时间被设置。
  - ACTION_TIMEZONE_CHANGED：时间区改变。
  - ACTION_BOOT_COMPLETED：系统完成启动后，一次广播。
  - ACTION_PACKAGE_ADDED：一个新应用包已经安装在设备上，数据包括包名(最新安装的包程序不能接收到这个广播)。
  - ACTION_PACKAGE_CHANGED：一个已存在的应用程序包已经改变，包括包名。
  - ACTION_PACKAGE_REMOVED：一个已存在的应用程序包已经从设备上移除，包括包名(正在被安装的包程序不能接收到这个广播)。
  - ACTION_UID_REMOVED：一个用户 ID 已经从系统中移除。
  - ACTION_PACKAGE_RESTARTED：用户重新开始一个包，包的所有进程将被杀死，所有与其联系的运行时间状态应该被移除，包括包名(重新开始包程序不能接收到这个广播)。
  - ACTION_PACKAGE_DATA_CLEARED：用户已经清除一个包的数据，包括包名(清除包程序不能接收到这个广播)。
  - ACTION_BATTERY_CHANGED：电池的充电状态、电荷级别改变，不能通过组件声明接收这个广播，只有通过 Context.registerReceiver()注册。

(3)　Data 描述了 Intent 要操作的数据 URI 和数据类型。有些动作需要对相应的数据进行处理，例如，对于动作 ACTION_EDIT 来说，它的数据可以为联系人、短信息等可编辑的 URI，而对于 ACTION_CALL 来说，它的数据可以是一个“tel://”格式的电话号码 URI。

正确设置 Intent 的数据对于 Android 寻找系统中匹配 Intent 请求的组件很重要。如果使用了 ACTION_CALL，但数据却设置成了“mail to://”格式的 URI，那么所期望的“启动打电话应用

程序”动作会因没有与之相对应的应用程序而不会被执行。所以每次使用 Intent 时，都应该留意与设置的 Action 相关的数据类型和格式。

(4) Category 是对被请求组件的额外描述信息。Android 也在 Intent 类中定义了一组静态字符串常量表示 Intent 不同的类别，常见的 Category 常量如下所示。

- ❑ CATEGORY_BROWSABLE：目标活动可以被浏览器安全的唤醒显示被一个链接所引用的数据，比如，一张图片或一条 E-mail 消息。
- ❑ CATEGORY_GADGET：这个活动可以被嵌入到充当配件宿主的另外的活动中。
- ❑ CATEGORY_HOME：这个活动将显示桌面，也就是用户开机后看到的第一个屏幕或者按 HOME 键时看到的屏幕。
- ❑ CATEGORY_LAUNCHER：这个活动可以是一个任务的初始活动并被列在应用程序启动器的顶层。
- ❑ CATEGORY_PREFERENCE：目标活动是一个选择面板。

8.2 拨 打 电 话

在本节的内容中，通过一个具体实例演示用 Intent 实现拨打电话功能的方法。

题　目	目　的	源码路径
演练 1	实战演练如何在应用程序中使用 Intent 实现拨打电话	“光盘:\daima\8\DiaPhone”文件夹

《秘籍》中说道：使用一个 Intent 打开电话拨号程序，Intent 的行为是 ACTION_DIAL，同时在 Intent 中传递被呼叫人的电话号码。程序的实现过程分为三个阶段，第一阶段，我们只完成向固定电话拨号的工作，用户不能自由输入希望通话的电话号码。第二阶段，进一步完善用户界面，让用户可以自由输入电话号码，然后再拨号。第三阶段，加入 IntentFilter，使得用户可以通过硬键盘拨号键启动拨号程序。

1. 基本的拨号程序

基本的拨号程序具体如下。

第 1 步：设置用户界面风格。新创建的项目中用户界面默认为 Hello Android 风格(只显示问候语字符串)，因此我们需要修改默认的用户界面，在用户界面中加入一个 Button 按钮。编辑 res/layout/main.xml 文件，删除<TextView>标签，加入新的<Button>标签。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
 <LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
 android:orientation="vertical"
 android:layout_width="fill_parent" android:layout_height="fill_parent"
 >
 <Button android:id = "@+id/button_id"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="@string/button"
```

```
 />
 </LinearLayout>
```

把 Button 的 id 设置为 button_id，同时将 Button 显示在界面上的文字设置为 res/string.xml/下的 Button，打开 res/string.xml，把 Button 的内容设置为“拨号”。

```
<?xml version="1.0" encoding="utf-8"?>
 <resources>
 <string name="button">拨号</string>
 <string name="app_name">TinyDiaPhone</string>
</resources>
```

第 2 步：创建 TinyDiaPhone 的 Activity，编写 TinyDiaPhone.java 代码。主要代码如下：

```
public class DiaPhone extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

第 3 步：定位“拨号”按钮。要加入对“拨号”按钮的响应，首先通过 findViewById()方法获得该按钮对象的引用。主要代码如下：

```
final Button button = (Button) findViewById(R.id.button_id);
```

第 4 步：加入对“拨号”按钮按键动作的响应。为“拨号”按钮对象调用 setOnClickListener()方法，设置单击事件监听器。主要代码如下：

```
final Button button = (Button) findViewById(R.id.button_id);
  button.setOnClickListener(new Button.OnClickListener() {
  @Override  public void onClick(View b) {
  //TODO 加入对按钮按下后的操作
  }
  });
```

第 5 步：创建 Intent 对象，用 Intent 启动新的 Activity。此项目希望在按钮被按下后发出一个启动系统自带拨号程序的 Intent，所以首先创建 Intent 对象。主要代码如下：

```
Intent <Intent_name> = new Intent(<ACTION>,<Data>)
```

在本例中，参数<ACTION>为 Intent.ACTION_DIAL，参数<Data>是希望传入的电话号码。

在 Android 中，传给 Intent 的数据用 URI 格式表示，因此需要使用 Uri.parse 方法将字符串格式的电话号码解析成 URI 格式。在上述演练中，用 tel：13800138000 表示我们想要呼叫的电话号码。那么，最终创建 Intent 的主要代码如下：

```
Intent I = new Intent(Intent.ACTION_DIAL,
  Uri.parse("tel://13800138000"));
```

创建 Intent 完毕后，就可以通过它告诉 Android 希望启动新的 Activity 了，主要代码如下：

```
startActivity(i);
```

至此，整个演练结束。运行后可以看到主界面如图 8-1 所示，这个界面的布局信息都在 main.xml 文件中，在一个 LinearLayout 当中数值排列了 5 个 Button。



图 8-1　主界面

2. 可输入电话号码的拨号程序

我们可以进一步完善前面的实例，例如，使用户可以输入电话号码。由于用户输入的内容可能不是一个有效的电话号码，所以程序还需要对用户输入的字符串进行判断，呼叫有效号码，如果是无效号码则提示用户重新输入。

第 1 步：修改用户界面，加入获取用户输入的 EditText 控件。在 Button 控件前加入一个 EditText 控件用于获取用户输入的电话号码，设置其 ID 引用名为 phonenumber_id。主要代码如下：

```
<EditText android:id = "@+id/phonenumber_id"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 />
 <Button android:id = "@+id/button_id"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content" android:text="@string/button"
 />
```

第 2 步：获得 EditText 对象的引用。主要代码如下：

```
final EditText phoneNumber =  (EditText)findViewById(R.id.phonenumber_id);
```

第 3 步：在回调方法 onClick 中加入对电话号码有效性的判断和处理。主要代码如下：

```
@Override  public void onClick(View b) {
  String callee = phoneNumber.getText().toString();
  if (PhoneNumberUtils.isGlobalPhoneNumber(callee)){
  Intent i = new Intent(Intent.ACTION_DIAL,  Uri.parse(tel"://"+ callee));
  startActivity(i);
  } else {
  Toast.makeText(TinyDiaPhone.this, R.string.notify_incorrect_phonenumber,
```

```
Toast.LENGTH LONG).show();
}
}
```

在上述代码中有如下几点需要说明。

(1) 判断电话号码的有效性可以使用 android.telephony.PhoneNumberUtils 包中的 isGlobalPhoneNumber 方法，Android 已经为我们准备了很多诸如此类的基本方法简化程序员的工作量，用好这些方法能够帮助我们轻松快速地完成工作。

(2) 使用 Toast 类实现无效电话号码的提示。读者可能会问，我怎么知道 Android SDK 提供了哪些机制帮助我实现不同的需求？一个好办法是查看模拟器中自带的 API Demos(API 演示)。

API Demos 已经分类为开发人员组织了很多实例，这些丰富多彩的实例是开发人员获取灵感的好地方。

第 4 步：编写文件 string.xml，具体实现代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="button">拨打电话</string>
   <string name="app_name">DiaPhone</string>
   <string name="text">输入电话</string>
   <string name="phonenumber">138XXXXXXXX</string>
   <string name="notify_incorrect_phonenumber">号码不正确，请重新输入！</string>
</resources>
```

至此，整个演练结束。执行后可以在框中输入要拨打的号码，输入完毕后按下“拨号”键后将看到用户界面切换到 Android 自带的拨号程序，同时你所拨打的电话号码会显示在屏幕上，如图 8-2 所示。

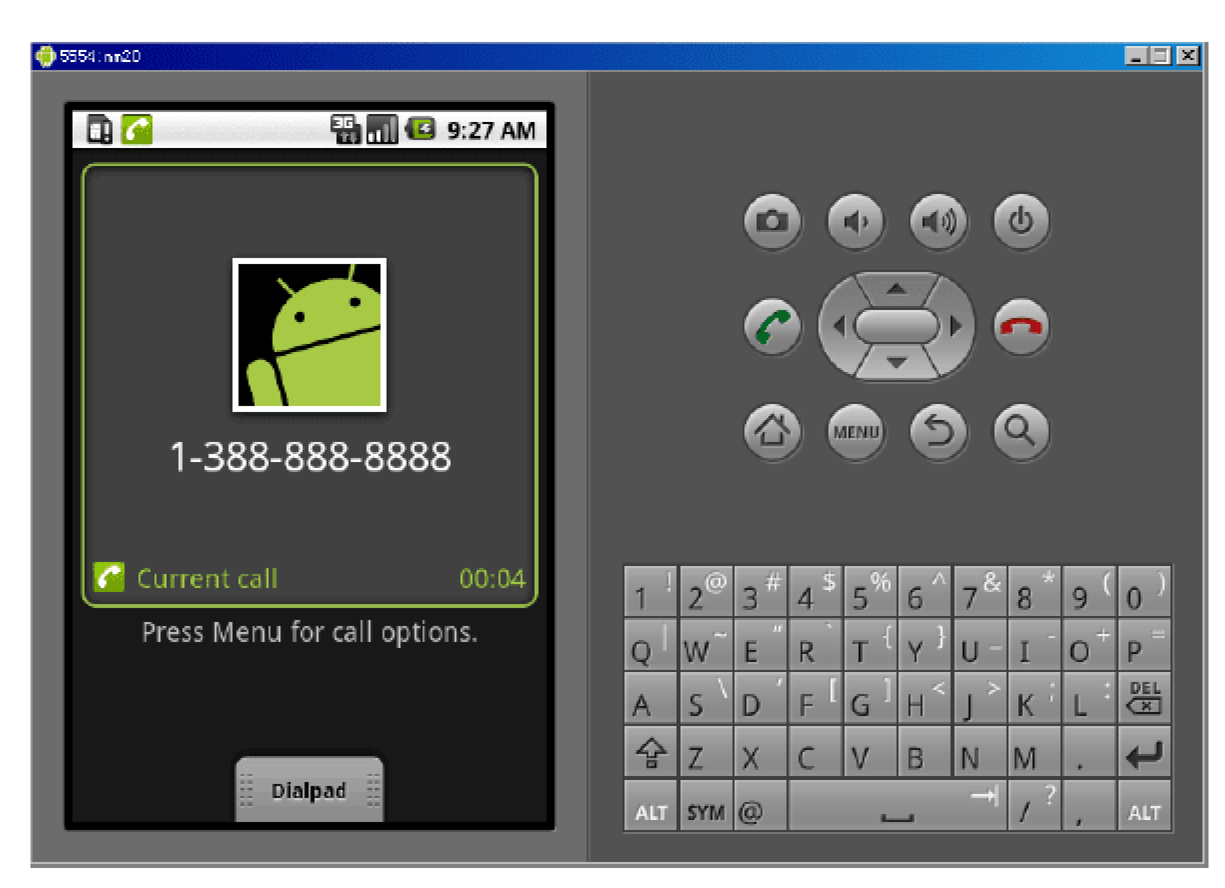


图 8-2　Android 自带拨号程序界面

3. IntentFilter 实现拨号处理

根据之前对 IntentFilter 的描述，硬件键盘的拨号键启动程序需要我们在 TinyDiaPhone 中加入一条新的 IntentFilter。在 AndroidManifest.xml 中关于 IntentFilter 的代码如下：

```
<intent-filter>
<action android:name="android.Intent.Action.MAIN" />
 <category android:name="android.Intent.Category.LAUNCHER" />
```

```
</intent-filter>
```

目前只有一条 IntentFilter，它的动作名称是 Action.MAIN，类别名称是 Category.LAUNCHER。正是有了这条 IntentFilter，TinyDiaPhone 的图标才出现在了应用程序的选择菜单里。为了新加入拨号键启动 TinyDiaPhone，增加的代码如下：

```
<intent-filter>
 <action android:name="android.Intent.Action.CALL_BUTTON"/ >
<category android:name="android.Intent.Category.DEFAULT" />
</intent-filter>
```

当按下键盘左下角绿色的拨号键时，系统会弹出一个窗口提醒用户，选择启动 TinyDiaPhone 还是选择 Android 自带的拨号程序。Android 自带拨号程序界面如图 8-3 所示。

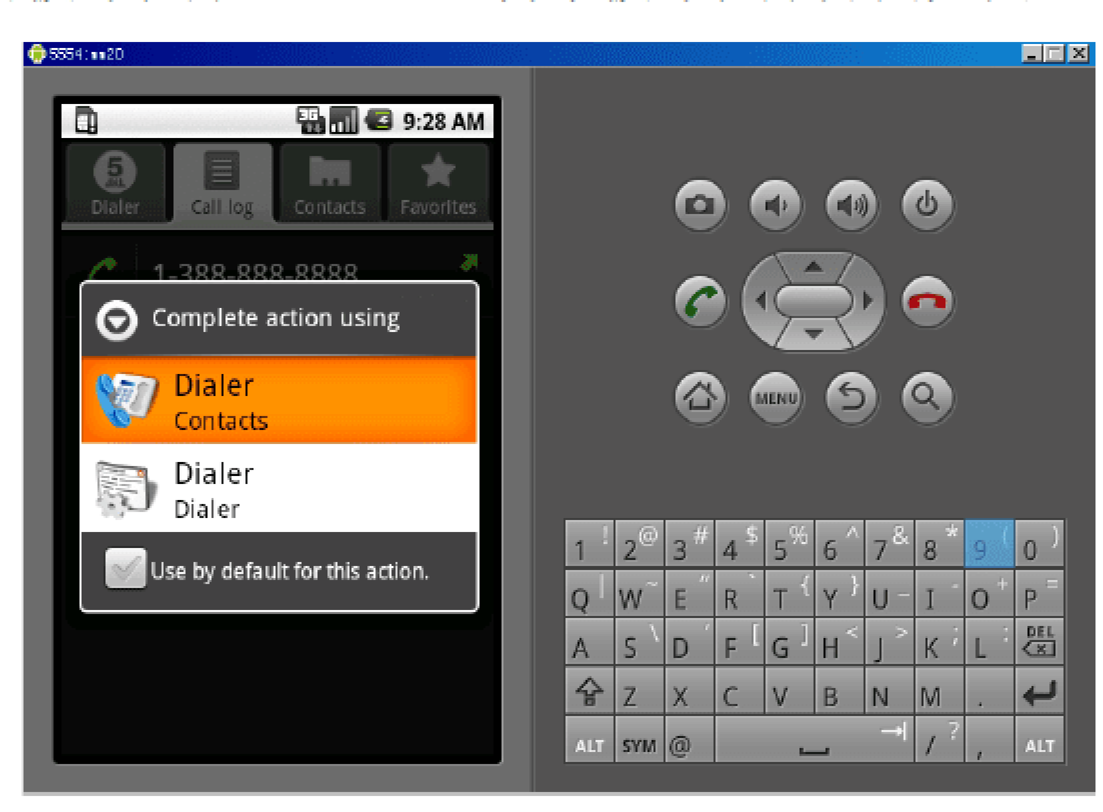


图 8-3　Android 自带拨号程序界面

此演练很好地说明了隐式 Intent 的用法。TinyDiaPhone 声明 IntentFilter 的行为是 ACTION.CALL_BUTTON，以后每次用户按下拨号键时，Android 系统都会将拨号键的意图和所有声明过 ACTION.CALL_BUTTON 的 IntentFilter 进行比较，然后将匹配的组件提供给用户进行选择。

8.3　双剑合璧大事记——发送短信

题　目	目　的	源码路径
演练 2	实战演练使用 SmsManager 类完成发送短信	“光盘:\daima\8\SMS”文件夹

秘籍中说道：同电话拨号程序一样，短信也是任何一款手机不可或缺的基本应用，是使用频率最高的程序之一。现在，我们再实现一个短信程序 TinySMS。此演练不是简单地使用 Intent 激活 Android 自带的短信程序，而是使用 SmsManager 类完成发送短信的功能。

8.3.1　创建 TinySMS 界面

第 1 步：修改用户界面 main.xml。具体代码如下所示：

```
…
    <TextView
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="对方电话"
        />
    <EditText
        android:id="@+id/txtPhoneNo"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="短信内容"
        />
    <EditText
        android:id="@+id/txtMessage"
        android:layout_width="fill_parent"
        android:layout_height="150px"
        android:gravity="top"
        />
    <Button
        android:id="@+id/btnSendSMS"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="发送"
        />
……
```

接下来编写文件 strings.xml。具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">SMS</string>
    <string name="app_name">发送短信</string>
</resources>
```

设计后的界面如图 8-4 所示。

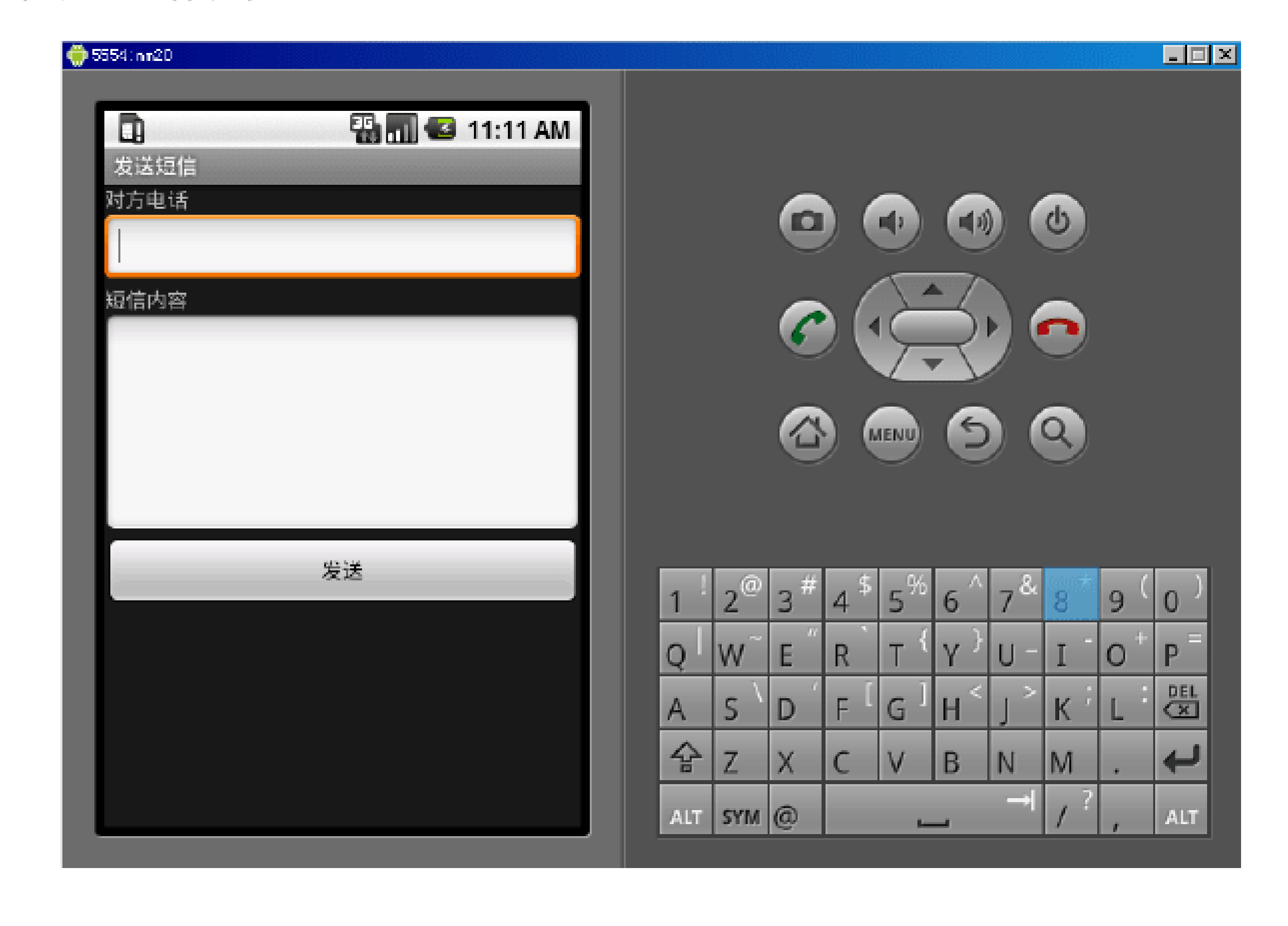


图 8-4　程序界面

8.3.2　设置权限

因为项目程序需要使用发送短信的功能，根据对 AndroidManifest.xml 的描述，在此需要在该文件中声明程序的权限。因此，这里需要加入 TinySMS 发送短信的权限声明。具体代码如下：

```
……
<uses-permission android:name="android.permission.SEND SMS" />
……
```

在上述代码中，“<uses-permission android:name="android.permission.SEND_SMS" />”是 TinySMS 发送短信的权限声明。

8.3.3　发送短信处理

当单击“发送短信”按钮后，通过事件处理的回调方法 onClick()实现发送短信的功能。具体代码如下所示：

```
        btnSendSMS.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View v)
            {
                String phoneNo = txtPhoneNo.getText().toString();
                String message = txtMessage.getText().toString();
                if (phoneNo.length()>0 && message.length()>0){
                  Log.v("ROGER", "will begin sendSMS");
                      sendSMS(phoneNo, message);
                }
                else
                    Toast.makeText(SMS.this,
                        "请重新输入",
                        Toast.LENGTH_LONG).show();
            }
        });
    }
```

TinySMS 并不是使用 Intent 激活 Android 自带的短信程序，而是直接使用了一个叫做 sendSMS 的方法。该方法的实现代码如下所示：

```
    private void sendSMS(String phoneNumber, String message)
    {
        PendingIntent pi = PendingIntent.getActivity(this, 0,
            new Intent(this, SMS.class), 0);
        Log.v("ROGER", "will init SMS Manager");
        SmsManager sms = SmsManager.getDefault();

        Log.v("ROGER", "will send SMS");
        sms.sendTextMessage(phoneNumber, null, message, pi, null);
    }
```

SmsManager 在 android.telephony.gsm.SmsManager 中定义，是实现用户管理短信应用的类。

在使用时，开发人员不用直接实例化 SmsManager 类，只需要调用静态方法 getDefault()即可获得 SmsManger 对象，方法 sendTextMessage()用于发送短信到指定号码。在上面这段代码中，使用了一个 PendingIntent 的对象，该对象指向 TinySMSActivity。因此当用户按下“发送短信”键之后，用户界面会重新回到 TinySMS 的初始界面。

在 Android 的模拟器中为短信或电话提供了非常方便的测试功能。用户只需要在 Windows 命令行中输入 emulator 再启动一个 Android 模拟器，这样就可以实现两个手机间的电话或者短信的测试。需要说明的是，每个模拟器左上角的数字代表了该模拟器的电话号码。

难道实际中的手机应用就这些吗？当然不是了！除了在上述实例中包含的内容以外，复杂的电话或短信应用可以参考 Android 的相关包，它们分别是 android.telephony 和 android.telephony.gsm。android.telephony 包中有如下类。

- ❑ CellLocation：设备位置的抽象类。
- ❑ PhoneNumberFormattingTextWather：用于监视一个 TextView 控件。
- ❑ PhoneNumberUtils：包含处理各种号码字符串的实用工具。
- ❑ PhoneStarteListener：监视手机电话状态变化的监听类。
- ❑ ServiceState：包括了电话状态和相关的服务信息。
- ❑ SMSManager：表示具体短信。
- ❑ TelephoneManager：提供对手机中电话服务信息的访问。

跟短信服务相关的类主要在包 android.telephony.gsm 中，包含有如下类。

- ❑ GSMCellLocation：GSM 手机的基站位置。
- ❑ GSMMessage：表示具体的短信。
- ❑ GSMManage：管理各种短信的操作。

SmsManager 是 android.telephony.gsm.SmsManager 中定义的用户管理短信应用的类。它的用法有点特殊，开发人员不用直接实例化 SmsManager 类，而只需要调用静态方法 getDefault()获得 SmsManger 对象，方法 sendTextMessage()用于发送短信到指定的号码。在上面这段代码中，我们使用了一个 PendingIntent 的对象，该对象指向一个 Activity 对象。因此，当用户按下“发送短信”键之后，用户界面会重新回到这个 Activity 的初始界面。



第 9 章　千里走单骑

Google 地图对大家来说应该不算陌生，它让我们体会到了高科技的奥妙。作为 google 旗下的产品之一，Android 中可以使用 Google 地图，地图功能是在 Google API 中应用的。在本章内容中，将详细讲解在 Android 中使用位置服务和地图 API 的基本流程。

9.1　实现 GPS 定位

Android 支持 GPS 和 google 网络地图，通常将各种不同的定位技术称为 LBS。LBS 是基于位置服务(Location Based Service)的简称，它是通过电信移动运营商的无线电通讯网络(如 GSM 网、CDMA 网)或外部定位方式(如 GPS)获取移动终端用户的位置信息(地理坐标)，在 GIS(Geographic Information System，地理信息系统)平台的支持下，为用户提供相应服务的一种增值业务。通过 GPS 定位技术，即使独自一人千里走单骑般的远行，也能保证不会迷路。

9.1.1　android.location 功能类

Android 支持地理定位服务的 API，该地理定位服务可以用来获取当前设备的地理位置。应用程序可以定时请求更新设备当前的地理定位信息。应用程序也可以借助一个 Intent 接收器来实现如下功能：以经纬度和半径划定一个区域，当设备出入该区域时，可以发出提醒信息。在下面的内容中，讲解 android.location 中和定位有关的功能类。

1. Google Map API

Android 也提供了一组访问 Google Map 的 API，借助 Google Map 定位 API，我们就能在地图上显示用户当前的地理位置。在 Android 中定义了一个名为 com.google.android.maps 的包，其中包含了一系列用于在 Google Map 上显示，控制和层叠信息的功能类，下面是该包中最重要的几个类。

- MapActivity：用于显示 Google Map 的 Activity 类，它需要连接底层网络。
- MapView：用于显示地图的 View 组件，它必须和 MapActivity 配合使用。
- MapController：用于控制地图的移动。

- ❑ Overlay：是一个可显示于地图之上可绘制的对象。
- ❑ GeoPoint：一个包含经纬度位置的对象。

2. Android Location API

在 Android Location API 中，包含了 Android 中关于定位功能的类。

- ❑ LocationManager：本类提供访问定位服务的功能，也提供获取最佳定位提供者的功能。另外，临近警报功能(前面所说的那种功能)也可以借助该类来实现。
- ❑ LocationProvider：该类是定位提供者的抽象类。定位提供者具备周期性报告设备地理位置的功能。
- ❑ LocationListener：提供定位信息发生改变时的回调功能。必须事先在定位管理器中注册监听器对象。
- ❑ Criteria：该类使得应用能够通过在 LocationProvider 中设置的属性来选择合适的定位提供者。

9.1.2 Android 定位的基本流程

在 Android 中实现定位处理的基本流程如下。

(1) 准备 Activity 类

本步骤的目标是使用 Google Map API 来显示地图，然后使用定位 API 来获取设备的当前定位信息以在 Google Map 上设置设备的当前位置，用户定位会随着用户的位置移动而发生改变。

首先我们需要一个继承了 MapActivity 的 Activity 类，例如下面的代码：

```
class MyGPSActivity extends MapActivity {
        …
         }
```

要成功引用 Google Map API，还必须先在 AndroidManifest.xml 中定义如下信息：

```
<uses-library android:name="com.google.android.maps" />
```

(2) 使用 MapView

要让地图显示的话，需要将 MapView 加入到应用中来。例如，在布局文件(main.xml)中加入如下代码：

```
<com.google.android.maps.MapView
            android:id="@+id/myGMap"
            android:layout_width="fill_parent"
            android:layout height="fill parent"
            android:enabled="true"
            android:clickable="true"
            android:apiKey="API_Key_String"
/>
```

另外，要使用 Google Map 服务，还需要一个 API Key。可以通过如下方式获取 API Key。

- ❑ 找到 USER_HOME\Local Settings\Application Data\Android 目录下的 debug.keystore 文件。
- ❑ 使用 Keytool 工具来生成认证信息(MD5)，使用如下命令行。

  keytool -list -alias androiddebugkey -keystore <path_to_debug_keystore>.keystore -storepass

android -keypass android

- 打开“Sign Up for the Android Maps API”页面，输入之前生成的认证信息(MD5)后将获取 API key。
- 用获取的 API key 替换上面 AndroidManifest.xml 配置文件中的“API_Key_String”。

注意： 上面获取 API Key 的介绍比较简单，后面将会通过一个具体实例来演示获取 API Key 的方法。

接下来继续补全 MyGPSActivity 类的代码，在此以使用 MapView 为例，具体的代码如下：

```
class MyGPSActivity extends MapActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
    //创建并初始化地图
      gMapView = (MapView) findViewById(R.id.myGMap);
        GeoPoint p = new GeoPoint((int) (lat * 1000000), (int) (long
          * 1000000));
         gMapView.setSatellite(true);
         mc = gMapView.getController();
         mc.setCenter(p);
         mc.setZoom(14);
    }
}
```

另外，如果要使用定位信息的话，必须设置一些权限，在 AndroidManifest.xml 中的具体代码如下：

```
<uses-permission
android:name="android.permission.INTERNET"></uses-permission>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"></uses-permission>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>
```

(3)　定位管理器

定位管理器可以使用 Context.getSystemService 方法，并传入 Context.LOCATION_SERVICE 参数来获取。具体的代码如下：

```
LocationManager lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

接下来需要将之前的 MyGPSActivity 作一些修改，让它实现一个 LocationListener 接口，使其能够监听定位信息的改变。修改的代码如下：

```
class MyGPSActivity extends MapActivity implements LocationListener {
…
public void onLocationChanged(Location location) {}
public void onProviderDisabled(String provider) {}
public void onProviderEnabled(String provider) {}
public void onStatusChanged(String provider, int status, Bundle extras) {}
protected boolean isRouteDisplayed() {
return false;
}
}
```

然后添加一些代码，对 LocationManager 进行一些初始化工作，并在它的 onCreate()方法中注册定位监听器。具体的代码如下：

```
@Override
public void onCreate(Bundle savedInstanceState) {
LocationManager lm = (LocationManager)getSystemService(Context.LOCATION_ SERVICE);
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000L, 500.0f, this);
        }
```

这样代码中的 onLocationChanged 方法就会在用户的位置发生 500 米距离的改变之后进行调用。这里默认使用的 LocationProvider 是 GPS(GPS_PROVIDER)，也可以根据你的需要，使用特定的 Criteria 对象调用 LocationManger 类的 getBestProvider 方法获取其他的 LocationProvider。可参考以下代码实现 onLocationChanged 方法。

```
  public void onLocationChanged(Location location) {
    if (location != null) {
    double lat = location.getLatitude();
    double lng = location.getLongitude();
     p = new GeoPoint((int) lat * 1000000, (int) lng * 1000000);
     mc.animateTo(p);
   }
}
```

通过上面的代码，获取了当前的新位置并更新地图上的位置显示。还可以为应用程序添加一些诸如缩放效果、地图标注、文本等功能。

(4) 添加缩放控件

在地图上添加缩放控件的代码如下：

```
//将缩放控件添加到地图上
ZoomControls zoomControls =  (ZoomControls) gMapView.getZoomControls();
zoomControls.setLayoutParams(new ViewGroup.LayoutParams(LayoutParams.WRAP_CONTENT,
LayoutParams.WRAP_CONTENT));
gMapView.addView(zoomControls);
gMapView.displayZoomControls(true);
```

(5) 添加 Map Overlay

这是最后一步，在地图上添加 Map Overlay。通过下面的代码可以定义一个 Overlay：

```
  class MyLocationOverlay extends com.google.android.maps.Overlay {
  @Override
   public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when)
{
  super.draw(canvas, mapView, shadow);
  Paint paint = new Paint();
  // 将经纬度转换成实际屏幕坐标
  Point myScreenCoords = new Point();
  mapView.getProjection().toPixels(p, myScreenCoords);
  paint.setStrokeWidth(1);
  paint.setARGB(255, 255, 255, 255);
  paint.setStyle(Paint.Style.STROKE);
```

```
    Bitmap bmp = BitmapFactory.decodeResource(getResources(), R.drawable.marker);
    canvas.drawBitmap(bmp, myScreenCoords.x, myScreenCoords.y, paint);
    canvas.drawText("how are you…", myScreenCoords.x, myScreenCoords.y, paint);
    return true;
  }
  }
```

上面的 Overlay 会在地图上显示一段文本，接下来可以把 Overlay 添加到地图上去，代码如下：

```
MyLocationOverlay myLocationOverlay = new MyLocationOverlay();
List<Overlay> list = gMapView.getOverlays();
list.add(myLocationOverlay);
```

9.1.3　GPS 定位应用实例

题　目	目　的	源码路径
演练 1	实战演练用 GPS 定位技术获取当前的位置信息	"光盘:\daima\9\GPSCurrentLocation"文件夹

用 GPS 定位技术获取当前位置信息的操作步骤如下。

第 1 步：在 AndroidManifest.xml 中添加 ACCESS_FINE_LOCATION 权限，具体代码如下：

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

第 2 步：在 onCreate(Bundle savedInstanceState)中获取当前位置信息，具体代码如下所示：

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        LocationManager locationManager;
        String serviceName = Context.LOCATION_SERVICE;
        locationManager = (LocationManager)getSystemService(serviceName);
        //String provider = LocationManager.GPS_PROVIDER;

        Criteria criteria = new Criteria();
        criteria.setAccuracy(Criteria.ACCURACY_FINE);
        criteria.setAltitudeRequired(false);
        criteria.setBearingRequired(false);
        criteria.setCostAllowed(true);
        criteria.setPowerRequirement(Criteria.POWER_LOW);
        String provider = locationManager.getBestProvider(criteria, true);

        Location location = locationManager.getLastKnownLocation(provider);
        updateWithNewLocation(location);
         /*每隔 1000ms 更新一次，并且不考虑位置的变化。*/
        locationManager.requestLocationUpdates(provider, 2000, 10,
             locationListener);
    }
```

在上述代码中，LocationManager 用于周期获得当前设备的一个类。要想获取 LocationManager 实例，需要调用 Context.getSystemService()方法并传入服务名 LOCATION_SERVICE("location")。

当创建 LocationManager 实例后，就可以通过 getLastKnownLocation()方法，将上一次 LocationManager 获得的有效位置信息以 Location 对象的形式返回。

第 3 步：定义方法 updateWithNewLocation(Location location)，用于更新显示用户界面，具体代码如下所示：

```
    private void updateWithNewLocation(Location location) {
     String latLongString;
     TextView myLocationText;
     myLocationText = (TextView)findViewById(R.id.myLocationText);
     if (location != null) {
     double lat = location.getLatitude();
     double lng = location.getLongitude();
     latLongString = "纬度:" + lat + "\n 经度:" + lng;
     } else {
     latLongString = "获取失败";
     }
     myLocationText.setText("当前位置是:\n" +
     latLongString);
    }
}
```

第 4 步：定义 LocationListener 对象 locationListener，当坐标改变时触发此对象。如果 Provider 传入相同的坐标则不会被触发，具体代码如下：

```
  private final LocationListener locationListener = new LocationListener() {
   public void onLocationChanged(Location location) {
   updateWithNewLocation(location);
   }
   public void onProviderDisabled(String provider){
   updateWithNewLocation(null);
   }
   public void onProviderEnabled(String provider){ }
   public void onStatusChanged(String provider, int status,
   Bundle extras){ }
  };
```

至此整个演练结束。因为模拟器上没有 GPS 设备，所以需要在 Eclipse 的 DDMS 工具中提供模拟的 GPS 数据。即依次单击 DDMS | Emulator Control 菜单命令，在弹出的对话框中找到 Location Control 选项，在打开对话框中输入坐标，完成后单击 Send 按钮，如图 9-1 所示。



图 9-1　设置坐标

因为用到了 Google API，所以要在项目中引入 Google API，右键单击项目选择 Properties，在弹出的对话框中选择 Google API 版本，如图 9-2 所示。这样模拟器运行后，会显示当前的坐标，执行结果如图 9-3 所示。

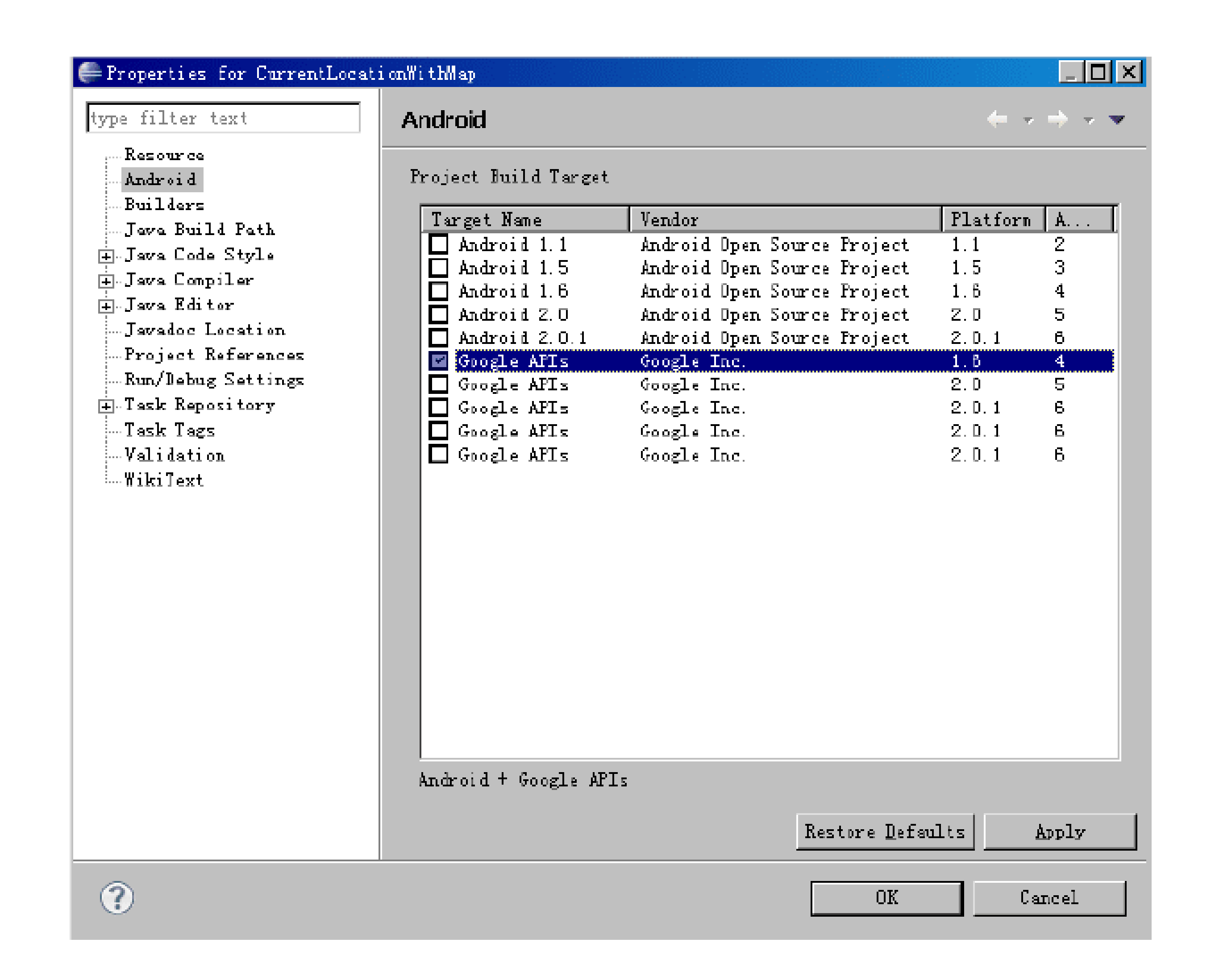


图 9-2　选择 Google API

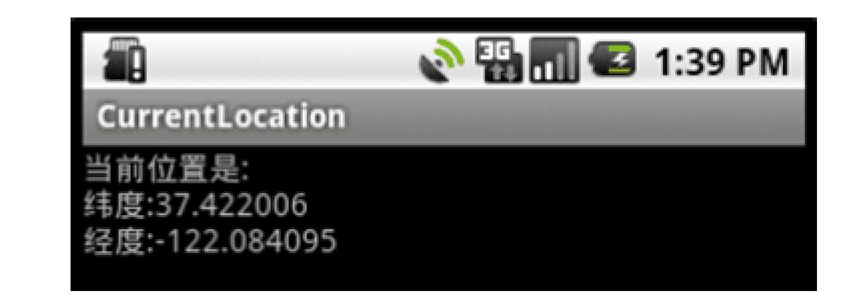


图 9-3　执行效果

9.1.4　LocationProvider 查询条件面面观

Android 中主要是通过 GPS 和网络来实现定位处理，即 GPS_PROVIDER 和 NETWORK_PROVIDER。在前面的实例中，显示指定了使用某种 LocationProvider 获取位置信息。其实，还可以用程序查询 LocationProvider 条件，Android 会根据查询条件帮助程序选择最合适的提供器。在 Criteria 类中提供了如下查询条件。

- ❑ 电池消耗，分为高、中或低。
- ❑ 运营商费用。
- ❑ 海拔。
- ❑ 速度。
- ❑ 方向。
- ❑ 位置解析精度，分为高或低。

例如，要求低位置解析精度、低电池消耗、不获取海拔高度、方向和速度的查询条件，允许运营商计算费用。具体代码如下所示：

```
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.PowerRequirment(Criteria.POWER_LOW);
criteria.setAltitudeRequired(false);
criteria.setBearingRequired(false);
criteria.setBearingRequired(Criteria.ACCURACY_FINE);
criteria.setBearingRequired(false);
criteria.setSpeedRequired(false);
criteria.setCostAllowed(true);
```

设置好上述查询条件后，可以用 getBestProvider 方法获取和查询条件最匹配的 LocationProvider。具体的代码如下所示：

```
String bestProvider = LocationProvider.getBestProvider(criteria,true);
```

如果有多个 LocationProvider 满足查询条件，那么位置解析度最高的那个提供器会最终被使用；如果没有一个匹配查询条件的，那么查询条件将变宽松，并按照如下顺序进行新一轮的查询，直到找到一个提供器为止。

- 电池的消耗。
- 位置的精度。
- 是否返回海拔、速度和方向。

9.2　及时获取当前位置

如果 GPS 的位置信息变化后，是不是也能够及时地显示新位置的信息到界面上呢？是不是只有退出系统，然后重新启动系统后才能得到最新的 GPS 信息呢？其实不然，我们完全可以通过编程来实现及时地获取位置信息。

9.2.1　介绍 Maps 库类

Maps 库中提供了十几个类，其中最常用的有 MapController、Mapview、MapActivity 等。

(1)　MapController

控制地图移动、伸缩，以某个 GPS 坐标为中心，控制 MapView 中的 View 组件，管理 Overlay，提供 View 的基本功能。使用多种地图模式，卫星模式、街景模式来查看 Google Map。

常用的方法有：animateTo(GeoPoint point)、setCenter(GeoPoint point)、setZoom(int zoomLevel)等。

(2)　MapView

MapView 是用来显示地图的 View，它派生自 android.view.ViewGroup。当 MapView 获得焦点时，可以控制地图的移动和缩放。

地图可以以不同的形式来显示出来，如街景模式、卫星模式等，通过 setSatellite(boolean) setTraffic(boolean)、setStreetView(boolean) 方法。

MapView 只能被 MapActivity 来创建，这是因为 MapView 需要通过后台的线程来连接网络或者文件系统，而这些线程要由 MapActivity 来管理。

需要特别说明的一点是，android 版本中，Map 的 zoom 采用了 built-in 机制，可以通过 setBuiltInZoomControls(boolean)来设置是否在地图上显示 zoom 控件。

常用方法有：getController()、getOverlays()、setSatellite(boolean)、setTraffic(boolean)、setStreetView(boolean)、setBuiltInZoomControls(boolean)等。

(3)　MapActivity

MapActivity 是一个抽象类，任何想要显示 MapView 的 Activity 都需要派生自 MapActivity，并且在其派生类的 onCreate()中，都要创建一个 MapView 实例，可以通过 MapViewConstructor(然后添加到 View 中 ViewGroup.addView(View))或者通过 layout XML 来创建。

(4)　Overlay

Overlay 是覆盖到 MapView 的最上层，可以扩展其 onDraw 接口，自定义在 MapView 中显示一些自己的东西。MapView 通过 MapView.getOverlays()对 Overlay 进行管理。

除了 Overlay 这个基类，Google 还扩展了如下两个比较有用的 Overlay。

- MylocationOverlay：集成了 Android.location 中接收当前坐标的接口，集成 SersorManager 中 CompassSensor 的接口。只需要 enableMyLocation()，enableCompass 就可以让我们的程序拥有实时的 MyLocation 以及 Compass 功能(Activity.onResume()中)。
- ItemlizedOverlay：管理一个 OverlayItem 链表，用图片等资源在地图上作风格相同的标记。

(5) Projection：MapView 中 GPS 坐标与设备坐标的转换(GeoPoint 和 Point)。

9.2.2　使用 LocationManager 及时监听当前的位置

LocationManager 支持监听器模式，通过调用 requestLocationUpdates()方法，能够为其设置一个位置监听器 LocationListener。同时 requestLocationUpdates()方法还需要指定要使用的位置服务类型、位置更新时间和最新位移，这样可以确保在满足用户需求的前提下最低的电量消耗。

例如，设置了更新位置信息的最小间隔为 2 秒，位移变化在 10 米以上。如果 GPS 位置超过 10 米，且时间间隔超过 2 秒时，LocationListener 的回调方法 onLocationChanged()就会被调用，应用程序可以通过 onLocationChanged 来反映位置信息的变化。具体代码如下：

```
public class CurrentLocationWithMap extends MapActivity {
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      LocationManager locationManager;
      String context = Context.LOCATION SERVICE;
      locationManager = (LocationManager)getSystemService(context);
      //String provider = LocationManager.GPS_PROVIDER;
      /*Location Provider 查询条件*/
      Criteria criteria = new Criteria();
      criteria.setAccuracy(Criteria.ACCURACY_FINE);
      criteria.setAltitudeRequired(false);
      criteria.setBearingRequired(false);
      criteria.setCostAllowed(true);
      criteria.setPowerRequirement(Criteria.POWER_LOW);
      String provider = locationManager.getBestProvider(criteria, true);

      Location location = locationManager.getLastKnownLocation(provider);
      updateWithNewLocation(location);
      /*设置更新位置信息的最小间隔为 2 秒，位移变化在 10 米以上。*/
      locationManager.requestLocationUpdates(provider, 2000, 10,
           locationListener);
   }
  /*Location 发生变化时被调用*/
  private final LocationListener locationListener = new LocationListener() {
   public void onLocationChanged(Location location) {
   updateWithNewLocation(location);
   }
   public void onProviderDisabled(String provider){
   updateWithNewLocation(null);
   }
   public void onProviderEnabled(String provider){ }
```

```
    public void onStatusChanged(String provider, int status,
    Bundle extras){ }
   };
   private void updateWithNewLocation(Location location) {
    String latLongString;
    TextView myLocationText;
    myLocationText = (TextView)findViewById(R.id.myLocationText);
    if (location != null) {
        double lat = location.getLatitude();
        double lng = location.getLongitude();
        latLongString = "纬度:" + lat + "\n经度:" + lng;

        ctrlMap.animateTo(new GeoPoint((int)(lat*1E6),(int)(lng*1E6)));
    } else {
        latLongString = "无法获取";
    }
    myLocationText.setText("位置是:\n" +
    latLongString);

   }
```

读者可以尝试将上述代码直接添加到 9.1.3 的实例中，执行后在 DDMS 中修改经度和维度，会发现显示界面中的内容也随之发生变化。

9.3 在 Android 系统中使用地图

在 Android 系统中可以直接使用 google 地图，可以用地图的形式显示位置信息。在本节内容中，将讲解在 Android 中使用 Google 地图的方法。

9.3.1 使用前的准备

Google 地图给人们的生活带来了极大的方便。例如，可以通过 Google 地图查找商户信息、查看地图和获取行车路线等。Android 平台也提供了一个 Map 包(com.google.android.maps)，通过其中的 MapView 就能够方便地利用 Google 地图的资源来进行编程。在使用前需要预先进行如下的设置。

(1) 添加 maps.jar

在 Android SDK 中，以 JAR 库的形式提供了和 Google Map 有关的 API，此 JAR 库位于“android-sdk-windows\add-ons\google_apis-4”目录下。要把 maps.jar 添加到项目中，可以在项目属性中的 Android 栏中指定使用包含 Google API 的 Target 作为项目的构建目标，如图 9-4 所示。

(2) 将地图嵌入到应用

通过使用 MapActivity 和 MapView 控件，可以轻松地将地图嵌入到应用程序当中。在此步骤中，需要将 Google API 添加到构建路径中。方法是在如图 9-4 所示的界面中选择 Java Build Path，然后在 Target 中选择勾选 Google API 选项，设置项目中包含 Google API，如图 9-5 所示。

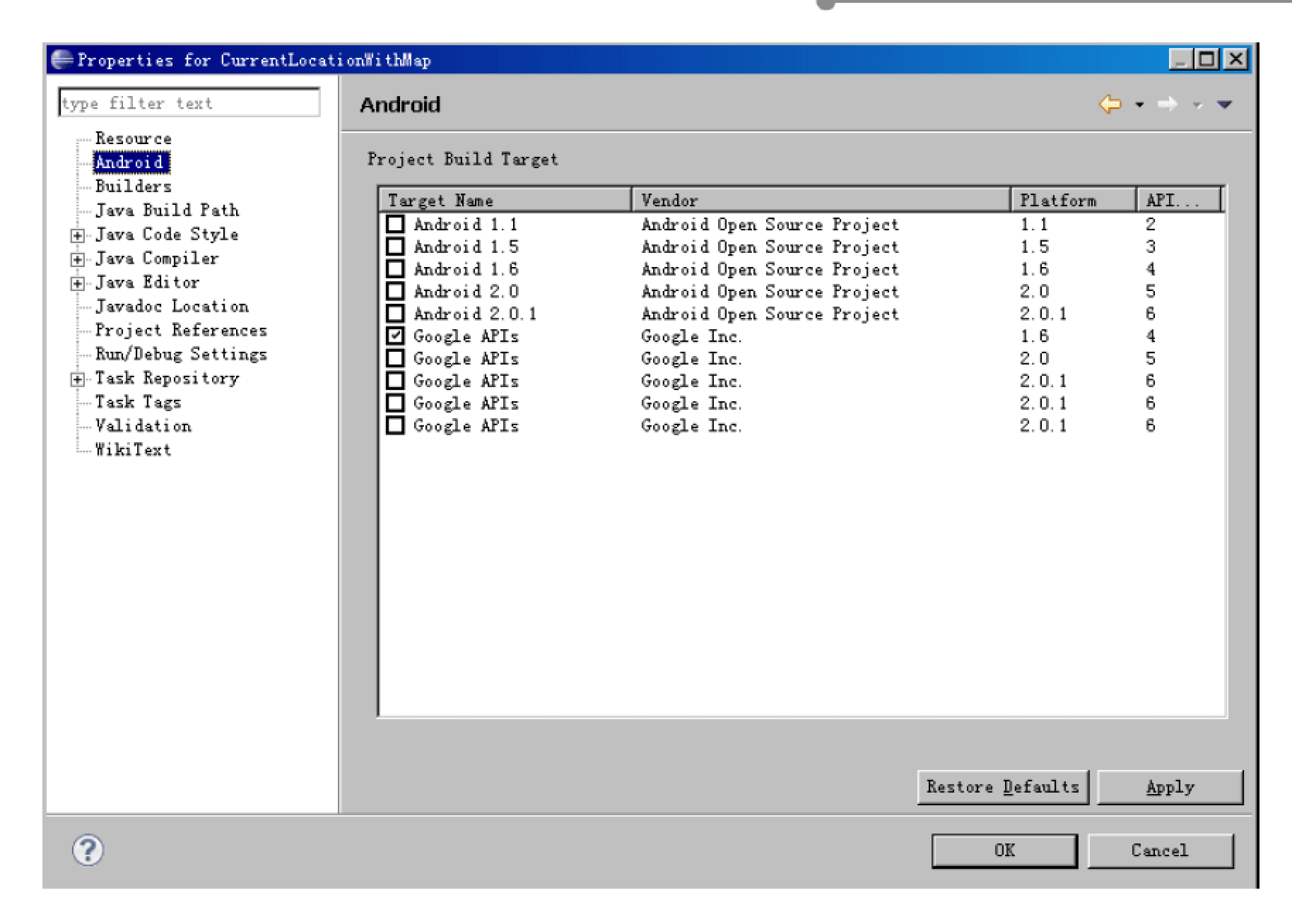


图 9-4　在项目中包含 Google API

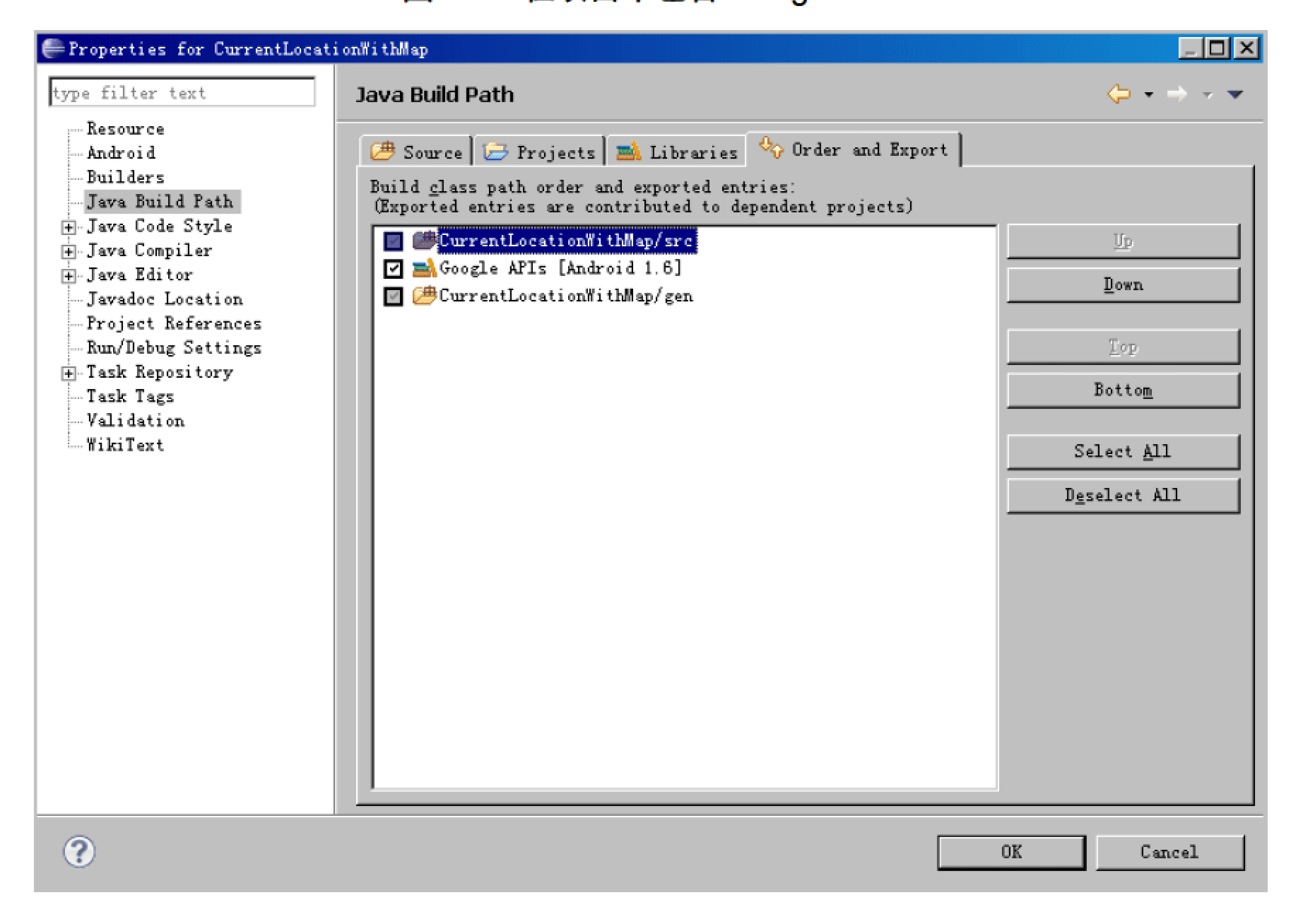


图 9-5　将 Google API 添加到构建路径

(3)　获取 Map API 密钥

在利用 MapView 之前，必须要先申请一个 Android Map API Key。具体步骤如下。

第 1 步：找到你的 debug.keystore 文件，通常位于如下目录：

C:\Documents and Settings\你的当前用户\Local Settings\Application Data\Android

第 2 步：获取 MD5 指纹，运行 cmd.exe，执行命令如下：

```
>keytool -list -alias androiddebugkey -keystore "debug.keystore 的路径" -storepass
android -keypass android
```

例如，笔者本人机器输入如下命令：

```
keytool -list -alias androiddebugkey -keystore "C:\Documents and Settings\
Administrator\android\debug.keystore" -storepass android -keypass android
```

此时系统会提示输入 keystore 密码，这时候输入 android，系统就会输出我们申请到的 MD5 认证指纹，如图 9-6 所示。

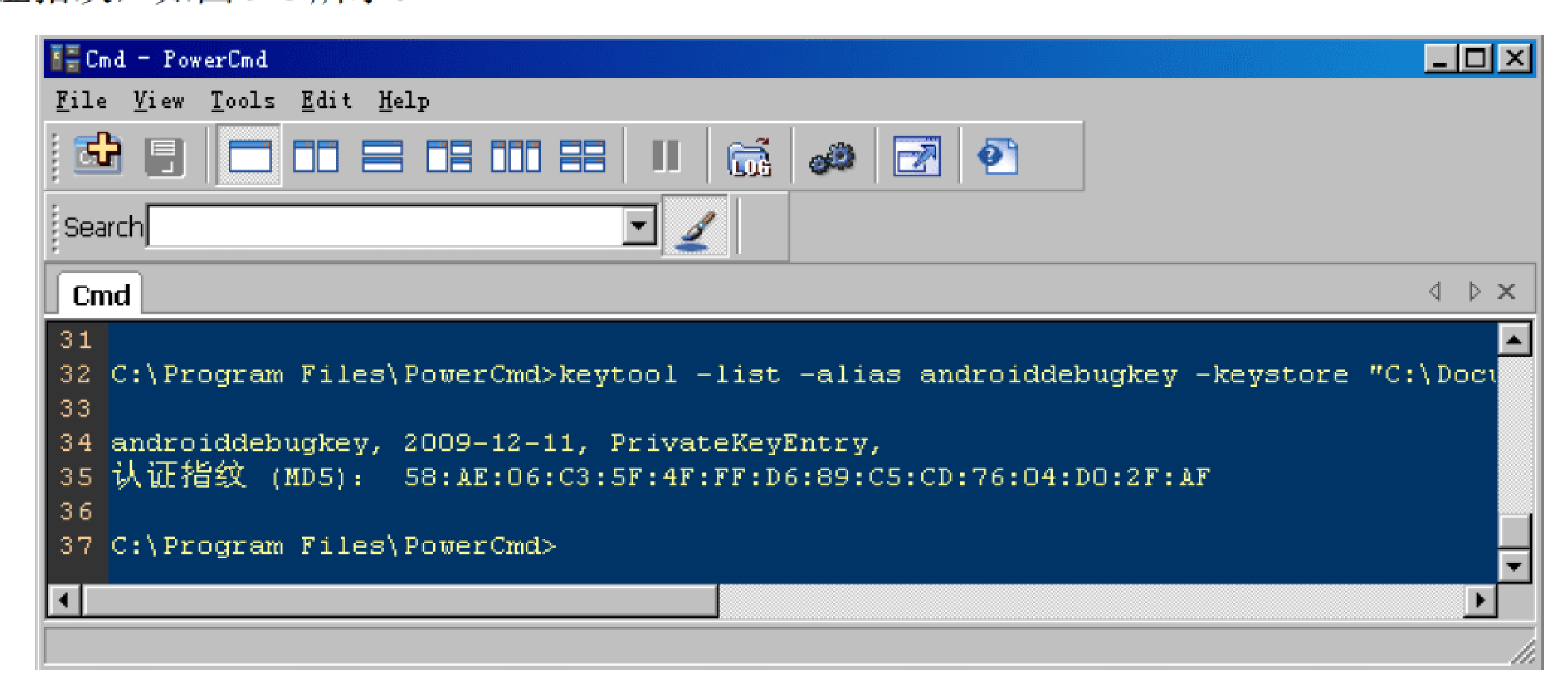


图 9-6　获取的认证指纹

注意：因为在 CMD 中不能直接复制、粘贴使用 CMD 命令，这样很影响编程效率，所以笔者使用了第三方软件 PowerCmd 来代替机器中自带的 CMD 工具。

第 3 步：申请 Android Map 的 API Key。

打开浏览器，输入网址：http://code.google.com/intl/zh-CN/android/maps-api-signup.html，打开页面如图 9-7 所示。

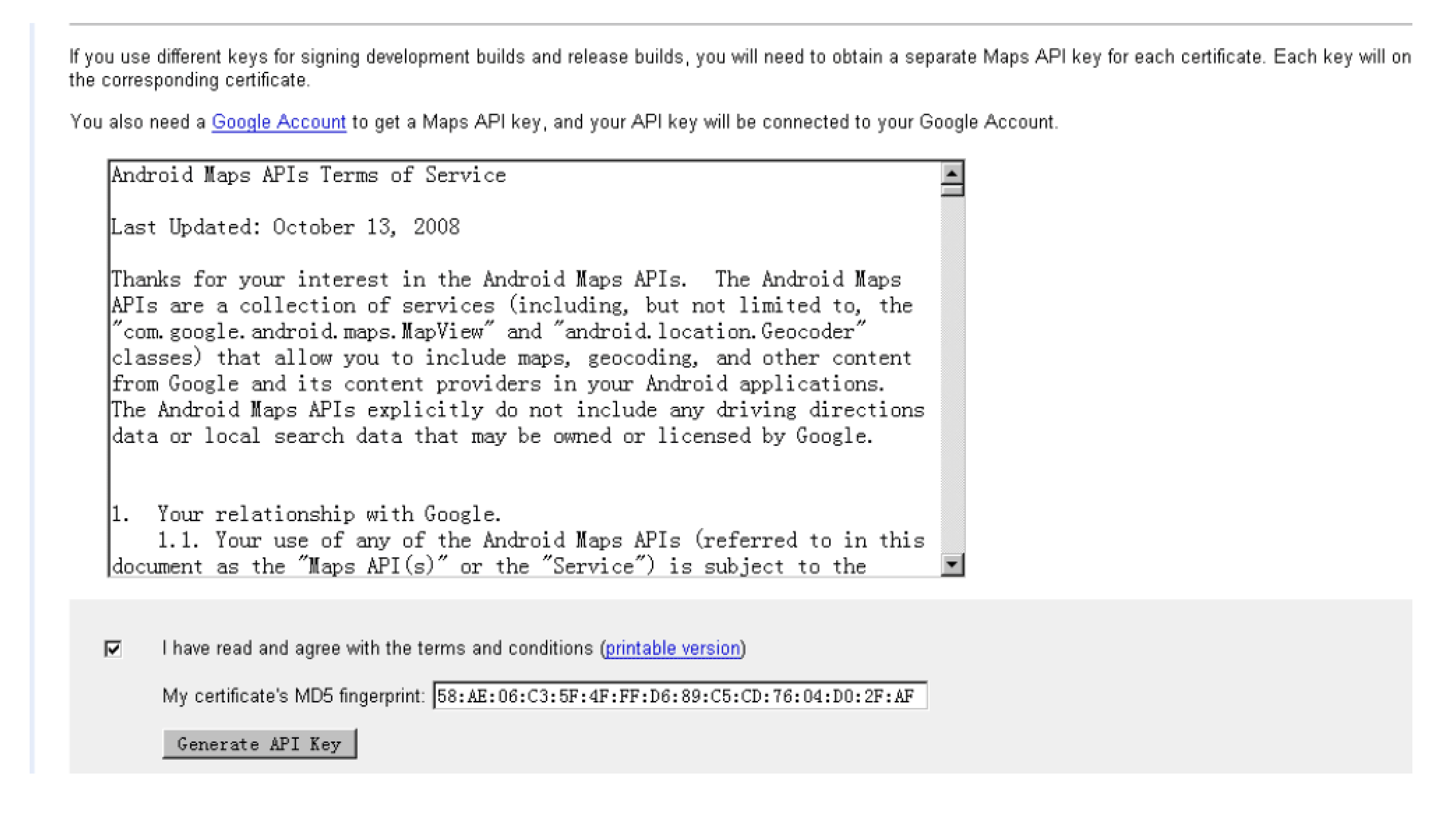


图 9-7　申请主页

在 google 的 Android Map API Key 申请页面上输入图 9-6 中得到的 MD5 认证指纹，按下 Generate API Key 按钮后即可转到下面的画面，得到了申请的 API Key，如图 9-8 所示。



图 9-8　申请 API Key

至此，成功地获取了一个 API Key，编程前的整个准备工作也完成了。

9.3.2　使用 Map API 密钥的基本流程

通过前面的讲解，我们已经申请到了一个 Android Map API Key，下面开始讲解使用 Map API 密钥实现编程的基本流程。

第 1 步：在 AndroidManifest.xml 中声明权限。

在 Android 系统中，如果程序执行需要读取到安全敏感的项目，那么必须在 AndroidManifest.xml 中声明相关权限请求，比如这个地图程序需要从网络读取相关数据。所以必须声明 android.permission.INTERNET 权限。具体方法是在 AndroidManifest.xml 中添加如下代码：

```
<uses-permission android:name="android.permission.INTERNET" />
```

另外，因为 maps 类不是 Android 启动的缺省类，所以还需要在文件 AndroidManifest.xml 的 Application 标签中声明要用 maps 类：

```
<uses-library android:name="com.google.android.maps" />
```

基本的 AndroidManifest.xml 文件代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   <application android:icon="@drawable/icon" android:label="@string/app_name">
    <uses-library android:name="com.google.android.maps" />
   </application>
 <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

第 2 步：在 main.xml 中完成布局。

假设要显示杭州的卫星地图并在地图上方有五个按钮，分别可以放大地图、缩小地图或者切换显示模式(卫星、交通、街景)。那么整个界面主要由两个部分组成，上面一排是五个按钮，下面是 Map View。

在 Android 中 LinearLayout 是可以互相嵌套的，在此可以把上面五个按钮放在一个子

LinearLayout 中(子 LinearLayout 的指定可以由 android:addStatesFromChildren="true"实现)，然后再把这个子 LinearLayout 加到外面的父 LinearLayout 中。具体实现的代码如下：

```
*为了简化篇幅，去掉一些不是重点说明的属性
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical" android:layout_width="fill_parent"
 android:layout_height="fill_parent">

 <LinearLayout android:layout_width="fill_parent"
  android:addStatesFromChildren="true"      /*说明是子 Layout
  android:gravity="center_vertical"         /*这个子 Layout 里边的按钮是横向排列
  >

  <Button android:id="@+id/ZoomOut"
   android:text="放大"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_marginTop="5dip"          /*下面的四个属性，指定了按钮的相对位置
   android:layout_marginLeft="30dip"
   android:layout_marginRight="5dip"
   android:layout_marginBottom="5dip"
   android:padding="5dip" />

  /*其余四个按钮省略
 </LinearLayout>
 <com.google.android.maps.MapView
  android:id="@+id/map"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:enabled="true"
  android:clickable="true"
  android:apiKey="在此输入上一节申请的 APIKEY"     /*必须加上上一节申请的 APIKEY
 />

</LinearLayout>
```

第 3 步：完成主 Java 程序代码，主文件中的这个类必须继承 MapActivity，然后在 onCreate()函数中创建基本对象。其主要代码如下：

```
public class Mapapp extends MapActivity {
public void onCreate(Bundle icicle) {
 //取得地图 View
  myMapView = (MapView) findViewById(R.id.map);
  //设置为卫星模式
  myMapView.setSatellite(true);
  //地图初始化的点:杭州
  GeoPoint p = new GeoPoint((int) (30.27 * 1000000),
    (int) (120.16 * 1000000));
  //取得地图 View 的控制
  MapController mc = myMapView.getController();
  //定位到杭州
  mc.animateTo(p);
```

```
 //设置初始化倍数
 mc.setZoom(DEFAULT_ZOOM_LEVEL);
}
```

接着，编写缩放按钮的处理代码。具体代码如下：

```
 btnZoomIn.setOnClickListener(new View.OnClickListener() {
  public void onClick(View view) {
    myMapView.getController().setZoom(myMapView.getZoomLevel() - 1);
 });
```

地图模式切换的实现代码如下：

```
  btnSatellite.setOnClickListener(new View.OnClickListener() {
  public void onClick(View view) {
   myMapView.setSatellite(true);                    //卫星模式为 True
   myMapView.setTraffic(false);                     //交通模式为 False
   myMapView.setStreetView(false);                  //街景模式为 False
  }
 });
```

至此，就完成了第一个使用 Map API 的应用程序。

9.3.3　用 Map API 密钥实现 Google 地图定位

题　目	目　的	源码路径
演练 2	实战演练使用 Map API 密钥实现 google 地图定位的基本流程	“光盘:\daima\9\UserCurrentLocationMap”文件夹

使用 Map API 密钥实现 Google 地图定位的操作流程如下。

第 1 步：在布局文件 main.xml 中插入了两个 Button，分别实现对地图的“放大”和“缩小”功能；然后，通过 ToggleButton 用于控制是否显示卫星地图；最后，设置申请的 APIKey。具体代码如下所示：

```
...
    <Button
        android:id="@+id/in"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="放大" />
    <Button
        android:id="@+id/out"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="缩小" />
<ToggleButton
    android:id="@+id/switchMap"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOff="卫星视图(关)"
```

```
    android:textOn="卫星视图(开)"/>
<com.google.android.maps.MapView
    android:id="@+id/myMapView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:clickable="true"
    android:apiKey="0by7ffx8jX0A_LWXeKCMTWAh8CqHAlqvzetFqjQ"
    />
…
```

第 2 步：在文件 AndroidManifest.xml 中，声明 android.permission.INTERNET 和 INTERNET 权限。具体代码如下所示：

```
…
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
…
```

第 3 步：编写主程序文件 CurrentLocationWithMap.java。

(1) 通过方法 onCreate 将 MapView 绘制到屏幕上。因为 MapView 只能继承 MapActivity 中的活动，所以必须用方法 onCreate 将 MapView 绘制到屏幕上，并同时覆盖方法 isRouteDisplayed()，它表示是否需要在地图上绘制导航线路。具体代码如下所示：

```
package com.UserCurrentLocationMap;
…
public class CurrentLocationWithMap extends MapActivity {
    MapView map;
    MapController ctrlMap;
    Button inBtn;
    Button outBtn;
    ToggleButton switchMap;
        @Override
    protected boolean isRouteDisplayed() {
        return false;
    }
```

(2) 定义方法 onCreate，首先引入主布局 main.xml 并通过方法 findViewById 获得 MapView 对象的引用，接着调用 getOverlays()方法获取其 Overlay 链表，并将构建好的 MyLocationOverlay 对象添加到链表中去。其中 MyLocationOverlay 对象调用的 enableMyLocation()方法表示尝试通过位置服务来获取当前的位置。具体代码如下所示：

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        map = (MapView)findViewById(R.id.myMapView);
        List<Overlay> overlays = map.getOverlays();
        MyLocationOverlay myLocation = new MyLocationOverlay(this,map);
        myLocation.enableMyLocation();
        overlays.add(myLocation);
```

还需要为“放大”和“缩小”这两个按钮设置处理程序，首先通过方法 getController()获取

MapView 的 MapController 对象，然后在“放大”和“缩小”两个按钮单击事件监听器的回放方法里，根据按钮的不同实现对 MapView 的缩放。具体代码如下所示：

```
ctrlMap = map.getController();
inBtn = (Button)findViewById(R.id.in);
outBtn = (Button)findViewById(R.id.out);
OnClickListener listener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.in:                               /*如果是缩放*/
            ctrlMap.zoomIn();
            break;
        case R.id.out:                              /*如果是放大*/
            ctrlMap.zoomOut();
            break;
        default:
            break;
        }
    }
};
inBtn.setOnClickListener(listener);
outBtn.setOnClickListener(listener);
```

(3)　通过方法 onCheckedChanged 来获取是否选择了 switchMap，如果选择了则显示卫星地图。首先通过方法 findViewById 获取对应 id 的 ToggleButton 对象的引用，然后调用 setOnCheckedChangeListener 方法，设置对事件监听器选中的事件进行处理。根据 ToggleButton 是否被选中，进而通过 setSatellite()方法启用或禁用卫星视图功能。具体代码如下所示：

```
switchMap = (ToggleButton)findViewById(R.id.switchMap);
switchMap.setOnCheckedChangeListener(new OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton cBtn, boolean isChecked) {
        if (isChecked == true) {
            map.setSatellite(true);
        } else {
            map.setSatellite(false);
        }
    }
});
```

(4)　通过 LocationManager 获取当前的位置，然后用 getBestProvider 方法来获取和查询条件，并设置更新位置信息的最小间隔为 2 秒，位移变化在 10 米以上。具体代码如下所示：

```
LocationManager locationManager;
String context = Context.LOCATION_SERVICE;
locationManager = (LocationManager)getSystemService(context);
//String provider = LocationManager.GPS_PROVIDER;

Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);
```

```
        criteria.setAltitudeRequired(false);
        criteria.setBearingRequired(false);
        criteria.setCostAllowed(true);
        criteria.setPowerRequirement(Criteria.POWER_LOW);
        String provider = locationManager.getBestProvider(criteria, true);

        Location location = locationManager.getLastKnownLocation(provider);
        updateWithNewLocation(location);
        locationManager.requestLocationUpdates(provider, 2000, 10,
             locationListener);
    }
```

(5) 设置回调方法何时被调用。具体代码如下所示：

```
  private final LocationListener locationListener = new LocationListener() {
   public void onLocationChanged(Location location) {
   updateWithNewLocation(location);
   }
   public void onProviderDisabled(String provider){
   updateWithNewLocation(null);
   }
   public void onProviderEnabled(String provider){ }
   public void onStatusChanged(String provider, int status,
   Bundle extras){ }
  };
```

(6) 定义方法 updateWithNewLocation(Location location)，用于显示地理信息和地图信息。具体代码如下所示：

```
   private void updateWithNewLocation(Location location) {
    String latLongString;
    TextView myLocationText;
    myLocationText = (TextView)findViewById(R.id.myLocationText);
    if (location != null) {
        double lat = location.getLatitude();
        double lng = location.getLongitude();
        latLongString = "纬度:" + lat + "\n经度:" + lng;

        ctrlMap.animateTo(new GeoPoint((int)(lat*1E6),(int)(lng*1E6)));
    } else {
        latLongString = "无法获取地理信息";
    }
    myLocationText.setText("您当前的位置是:\n" +
    latLongString);

   }
}
```

至此，整个演练结束，在图 9-9 中选定一个经度和纬度位置后，可以显示此位置的定位信息，并且定位信息分别以文字和地图的形式显示出来，如图 9-10 所示。

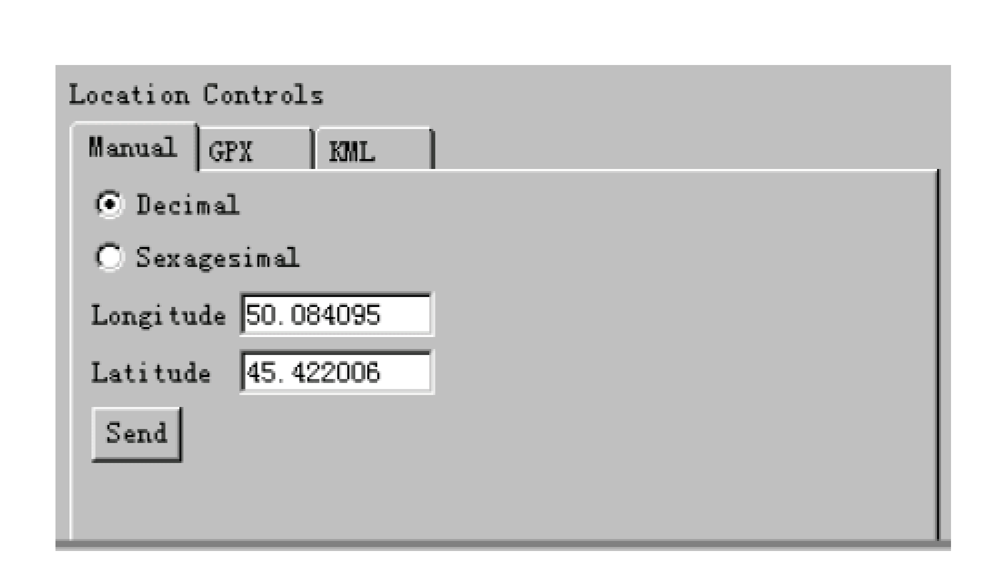


图 9-9　指定位置

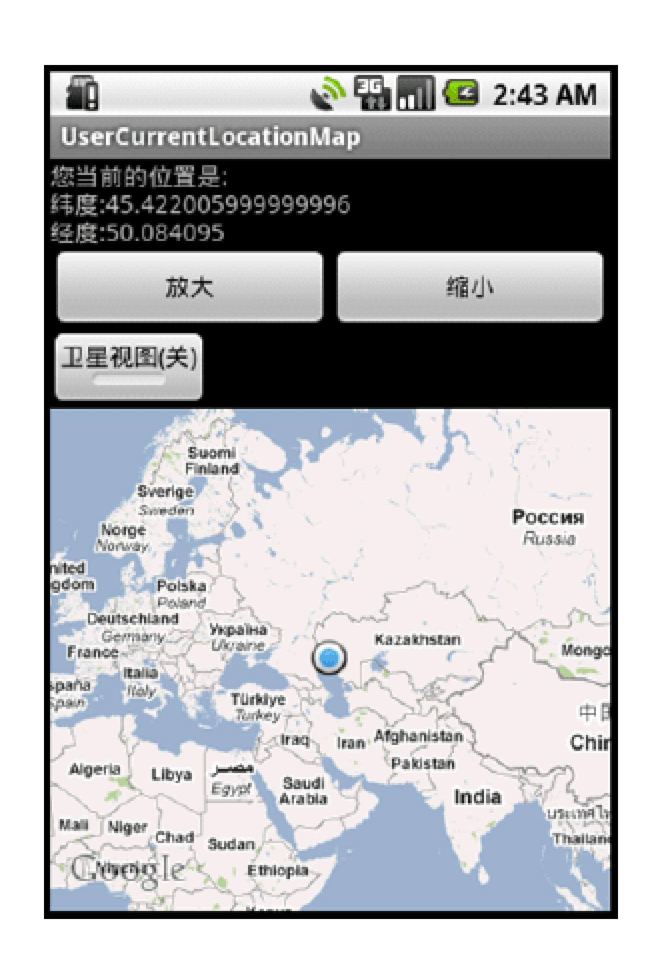


图 9-10　显示定位信息

单击“放大”和“缩小”按钮后，能控制地图的大小显示，放大后的效果如图 9-11 所示。打开卫星视图后，可以显示此位置范围对应的卫星地图，如图 9-12 所示。



图 9-11　放大后的效果



图 9-12　卫星地图

GPS 定位和 Google 地图一直是智能手机中的核心应用之一，所以本章的内容十分重要。请读者仔细学习本章的知识，只有这样才能掌握技术的精髓。



第三篇　实践闯关篇

第 10 章　第一关：交互式通信

作为一个移动手机设备，当然需要具备交互通信的功能。手机中的交互通信方式有多种，例如，常见的通话、短信、邮件和蓝牙等。在本章的内容中，将通过具体的实例来详细讲解 Android 中交互式通信应用的具体实现流程。

10.1　武 林 大 会

武林大会，就是手机界的奥林匹克运动会。手机联盟盟主特意号召群雄于 2011 年 12 月 24 日在兴云庄召开武林大会。大会的主题是和谐并举办一个闯关游戏，让各路群雄展现自己的精湛技艺。我作为一名安卓派的刚入门弟子，奉师命代表安卓派来参加闯关游戏。比赛之前，师傅给了我一本秘籍心法《Android SDK 4.0 密录》，说在大会上能给我带来好运。因为这次关系到师门的荣誉，于是我暗下决心，一定不能辜负师傅的期望，好好表现。

10.2　TextView 三维一体

题　目	目　的	源码路径
题目 1	提供一个文本输入框，当用户输入电话号码后，可以进行拨号处理；当输入一个网址后，可以登录到这个地址；当输入邮箱后，就可以发送邮件	“光盘:\daima\10\example1”文件夹

10.2.1 我的想法

经过权衡之后，我决定用 Linkify 对象实现上述功能。Linkify 可以让系统动态获取数据，并作出一个判断，判断是电话、邮箱还是网址，并随之作出对应的处理。通过 TextView 和 EditText 交互，就可以在此显示自己输入的数据值。具体实现上，只需重写 EditText.setOnkeyListener() 方法即可实现。

10.2.2 具体实现

编写的主程序文件是 example1.java。主要代码如下：

```
public class example1 extends Activity
{
  private TextView mTextView01;
  private EditText mEditText01;

    /** Called when the activity is first created. */
    @Override
  public void onCreate(Bundle savedInstanceState)
    {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mTextView01 = (TextView)findViewById(R.id.myTextView1);
    mEditText01 = (EditText)findViewById(R.id.myEditText1);

    mEditText01.setOnKeyListener(new EditText.OnKeyListener()
    {
      @Override
      public boolean onKey(View arg0, int arg1, KeyEvent arg2)
      {
        // TODO Auto-generated method stub
        mTextView01.setText(mEditText01.getText());
        /*判断输入的类型是何种，并与系统连接*/
       Linkify.addLinks(mTextView01,Linkify.WEB_URLS|Linkify.
           EMAIL_ADDRESSES|Linkify.PHONE_NUMBERS);
        return false;
      }
    });
    }
}
```

在上述代码中，设置了 Linkify 的处理事件，Linkify 是在创建 TextView 之后设置的，否则设置的 RE 不会起效果。同时还设置了 mTextView01 拥有自动链接功能，这样就能来到指定的页面。

执行后的效果如图 10-1 所示，用户可以输入电话号码、邮箱地址或网址，输入后可显示相应的操作。例如，输入 www.163.com 后，会在下面自动显示输入的网址，如图 10-2 所示；当单击网址后会来到相应的页面，如图 10-3 所示。

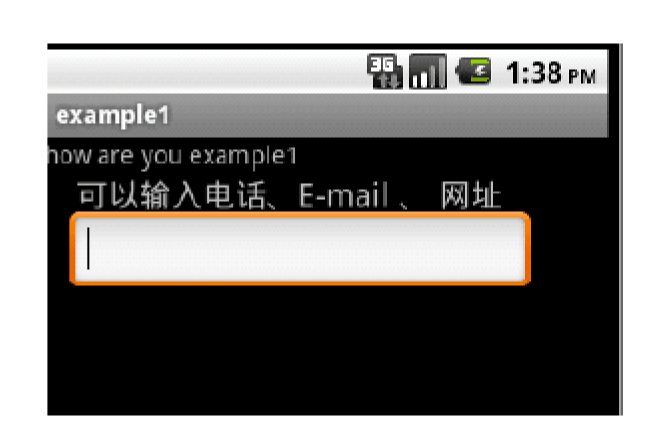


图 10-1　执行效果

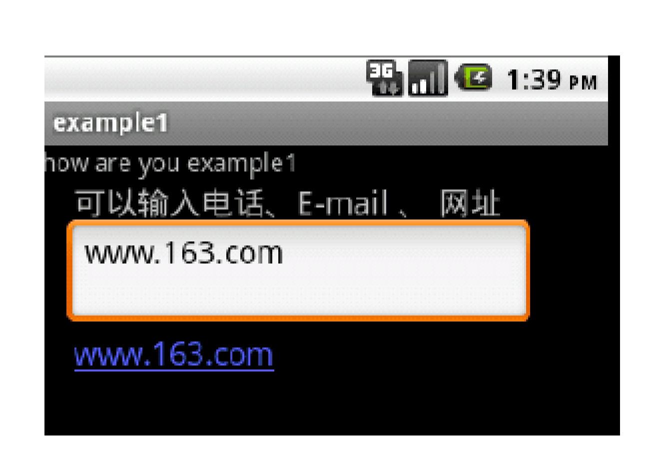


图 10-2　自动显示输入数据



图 10-3　来到指定网址

10.3　很熟悉的拨打电话

题　目	目　的	源码路径
题目 2	实现一个基本的手机拨打电话的过程	“光盘:\daima\10\example2”文件夹

10.3.1　我的想法

手机拨打电话的过程如下：先使用了一个 EditText 用于获取输入的电话号码，当单击 Button 后执行拨打电话程序，并且通过自定义的 isPhoneNumberValid(String phoneNumber)来确保用户输入的是合法的电话号码。在具体拨打电话时，首先要在 AndiroidManifest 中添加 uses-permission，这样就实现了对拨打电话的声明；然后通过自定义的 Intent 对象，通过“ACTION_CALL”键和 Uri.parse()方法将用户输入的电话号码写入；最后，通过 startActivity 方法即可完成拨打电话的功能。

10.3.2　具体实现

编写的主程序文件是 example2.java，下面开始讲解其具体的实现代码。

(1)　引用类和对象，声明 Button 与 EditText 对象名称，具体代码如下所示：

```
public class example2 extends Activity
{
 /*声明 Button 与 EditText 对象名称*/
  private Button mButton1;
```

```
private EditText mEditText1;
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  /*通过 findViewById 构造器来构造 EditText 与 Button 对象*/
  mEditText1 = (EditText) findViewById(R.id.myEditText1);
  mButton1 = (Button) findViewById(R.id.myButton1);
```

(2) 设置 Button 对象的 OnClickListener 来聆听 OnClick 事件。具体代码如下所示：

```
  mButton1.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View v)
    {
      try
      {
        /*取得 EditText 中用户输入的字符串*/
        String strInput = mEditText1.getText().toString();
        if (isPhoneNumberValid(strInput)==true)
        {
          /*建构一个新的 Intent
          运行 action.CALL 的常数与通过 Uri 将字符串带入*/
          Intent myIntentDial = new  Intent
          (
            "android.intent.action.CALL",
            Uri.parse("tel:"+strInput)
          );
          /*在 startActivity()方法中
          带入自定义的 Intent 对象以运行拨打电话的工作 */
          startActivity(myIntentDial);
          mEditText1.setText("");
        }
        else
        {
          mEditText1.setText("");
          Toast.makeText(
          example2.this, "输入的电话格式不符",
          Toast.LENGTH_LONG).show();
        }
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
    }
  });
}
```

(3) 定义 isPhoneNumberValid(String phoneNumber)方法，检查字符串是否为电话号码的方法，并返回 true or false 的判断值。具体代码如下所示：

```
public static boolean isPhoneNumberValid(String phoneNumber)
{
  boolean isValid = false;
  String expression = "^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{5})$";
  String expression2 ="^\\(?(\\d{3})\\)?[- ]?(\\d{4})[- ]?(\\d{4})$";
  CharSequence inputStr = phoneNumber;
  /*创建 Pattern*/
  Pattern pattern = Pattern.compile(expression);
  /*将 Pattern 以参数传入 Matcher 作 Regular expression*/
  Matcher matcher = pattern.matcher(inputStr);
  /*创建 Pattern2*/
  Pattern pattern2 =Pattern.compile(expression2);
  /*将 Pattern2 以参数传入 Matcher2 作 Regular expression*/
  Matcher matcher2= pattern2.matcher(inputStr);
  if(matcher.matches()||matcher2.matches())
  {
    isValid = true;
  }
  return isValid;
 }
}
```

程序执行后的效果如图 10-4 所示；如果输入的电话号码不规范则输出对应的提示，如图 10-5 所示；当输入规范的号码并单击“开始拨打”按钮后，会实现拨号处理并显示拨打界面，如图 10-6 所示。



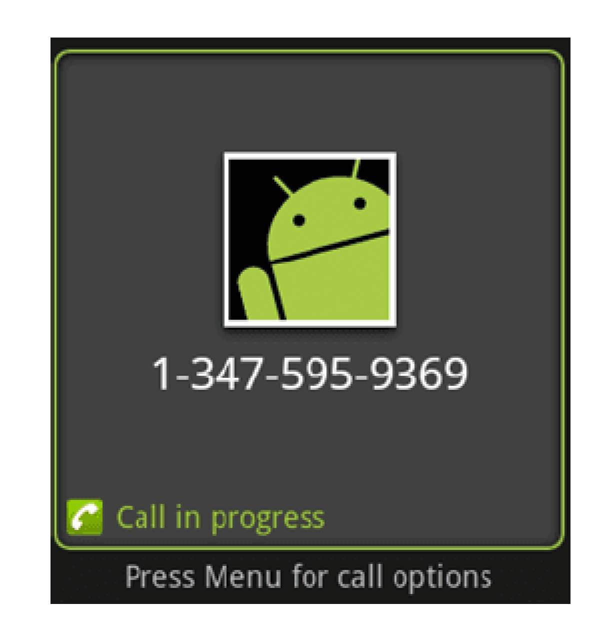


图 10-4　执行效果　　图 10-5　输出对应的提示　　图 10-6　拨打界面

10.4　新潮的 Email 程序

题　目	目　的	源码路径
题目 3	实现邮件发送功能	“光盘:\daima\10\example3”文件夹

10.4.1 我的想法

这个题目够新潮，原来没有接触过。幸亏有师傅给我的《Android SDK 4.0 密录》，上面提示我使用 Android.content.Intent.ACTION_SEND 的参数，可以通过手机实现收发 E-mail 服务的功能。其邮件的收发过程是通过 Android 内置的 Gmail 程序实现的，而并不是使用 SMTP 的 Protocol 实现的。为了确保邮件能够发出，必须在收件人字段上输入标准的邮件地址格式，如果格式不规范，则发送按钮处于不可用状态。

10.4.2 具体实现

编写的主程序文件是 example3.java，下面开始讲解其具体实现代码。

(1) 声明 EditTex、Button 和 String 变量，分别用于输入邮箱地址、邮件主体、副本和邮件内容。具体代码如下所示：

```
public class example3 extends Activity
{
  /*声明四个 EditText、一个 Button 以及四个 String 变量*/
  private EditText mEditText01;
  private EditText mEditText02;
  private EditText mEditText03;
  private EditText mEditText04;
  private Button mButton01;
  private String[] strEmailReciver;
  private String strEmailSubject;
  private String[] strEmailCc;
  private String strEmailBody ;
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /*通过 findViewById 构造器来建构 Button 对象*/
    mButton01 = (Button)findViewById(R.id.myButton1);
    /*通过 findViewById 构造器来构造所有 EditText 对象*/
    mButton01.setEnabled(false);
    /*设置 OnKeyListener，当 key 事件发生时进行反应*/
    mEditText01 = (EditText)findViewById(R.id.myEditText1);
    mEditText02 = (EditText)findViewById(R.id.myEditText2);
    mEditText03 = (EditText)findViewById(R.id.myEditText3);
    mEditText04 = (EditText)findViewById(R.id.myEditText4);
```

(2) 定义 setOnKeyListener 方法，如果用户输入为正规 E-mail 文字，则按钮不可用，反之则按钮可用。具体代码如下所示：

```
    /*若用户输入为正规 E-mail 文字，则 enable 按钮，反之则 disable 按钮*/
    mEditText01.setOnKeyListener(new EditText.OnKeyListener()
    {
```

```
    @Override
    public boolean onKey(View v, int keyCode, KeyEvent event)
    {
      // TODO Auto-generated method stub
      /*如果是邮件地址格式，则按钮可按下*/
      if(isEmail(mEditText01.getText().toString()))
      {
        mButton01.setEnabled(true);
      }
      else
      {
        mButton01.setEnabled(false);
      }
      return false;
    }
  });
```

(3) 定义 onClickListener 响应按钮，当单击按钮后实现邮件发送处理。具体代码如下所示：

```
  /*定义 onClickListener 响应按钮*/
  mButton01.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View v)
    {
      // TODO Auto-generated method stub
      Intent mEmailIntent = new Intent(android.content.Intent.ACTION_SEND);
      mEmailIntent.setType("plain/text");
      strEmailReciver = new String[]{mEditText01.getText().toString()};
      strEmailCc = new String[]{mEditText02.getText().toString()};
      strEmailSubject = mEditText03.getText().toString();
      strEmailBody = mEditText010.getText().toString();
      mEmailIntent.putExtra(android.content.Intent.EXTRA_EMAIL,
      strEmailReciver);
      mEmailIntent.putExtra(android.content.Intent.EXTRA CC, strEmailCc);
      mEmailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT,
      strEmailSubject);
      mEmailIntent.putExtra(android.content.Intent.EXTRA_TEXT,
      strEmailBody);
      startActivity(Intent.createChooser(mEmailIntent,
      getResources().getString(R.string.str_message)));
    }
  });
}
```

(4) 定义 isEmail(String strEmail)方法，检查是否是规范的邮件地址格式。具体代码如下所示：

```
 public static boolean isEmail(String strEmail)
 {
   String strPattern = "^[a-zA-Z][\\w\\.-]*[a-zA-Z0-9]@[a-zA-Z0-9][\\w\\.-]
*[a-zA-Z0-9]\\.[a-zA-Z][a-zA-Z\\.]*[a-zA-Z]$";
   Pattern p = Pattern.compile(strPattern);
```

```
    Matcher m = p.matcher(strEmail);
    return m.matches();
  }
}
```

至此整个展示结束，执行后的效果如图 10-7 所示，输入手机号码，编写短信内容后，单击“发送”按钮后即可完成短信发送功能，系统会提示成功信息。

注意：Android 模拟器中不会内置 Gmail Client 端程序，所以当使用本实例发送邮件后，会显示“No applications can perform this action”的提示，如图 10-8 所示。但是在现实手机设备上，如果运行本实例程序，会调用 Gmail 程序成功实现邮件发送。



图 10-7　初始效果



图 10-8　发送成功

10.5　震动你的心扉

盟主说：人的一生不能总是平平淡淡，喜怒哀乐、忧伤和高兴并重。但是生命中总会有一些事情和人能震动你的心扉，比如友情和爱情。接下来的一个题目很简单，实现一部手机震动效果。各位参赛朋友们，我们都知道手机除了基本的通话和短信外，震动也是另外一个极为重要的功能。通过手机震动，能够帮助我们及时感知打来的电话或发来的短信。

题　目	目　的	源码路径
题目 4	实现手机震动功能	“光盘:\daima\10\example4”文件夹

10.5.1　我的想法

《Android SDK 4.0 密录》提示我：使用 Android 中的震动事件 Vibratior 时，需要设置震动的时间长短和周期，并且设置单位是毫秒。如果要建立手机震动，则必须建立 Vibratior 对象，并通过调用 Vibrate 来实现震动目的。在 Vibratior 构造器中有四个参数，前三个用于设置震动大小，最后那个用于设置震动持续时间。我准备展示两种振动方式：一直持续和只震动一次。

10.5.2　具体实现

编写的主程序文件是 example4.java，下面开始讲解其具体实现代码。

(1)　设置 ToggleButton 的对象，检测 ToggleButton 是否被启动，如果按下 ON 按钮则启动震动模式，如果单击 OFF 按钮则关闭震动模式。具体代码如下所示：

```
public class example4 extends Activity
{
  private Vibrator mVibrator01;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /*设置 ToggleButton 的对象*/
    mVibrator01 = ( Vibrator )getApplication().getSystemService
    (Service.VIBRATOR_SERVICE);
    final ToggleButton mtogglebutton1 =
    (ToggleButton) findViewById(R.id.myTogglebutton1);
    final ToggleButton mtogglebutton2 =
    (ToggleButton) findViewById(R.id.myTogglebutton2);
    final ToggleButton mtogglebutton3 =
    (ToggleButton) findViewById(R.id.myTogglebutton3);
```

(2)　设置短震动模式，通过 mVibrator01.vibrate(new long[]{100,10,100,1000},-1)设置了此模式下的震动周期。具体代码如下所示：

```
    /* 短震动 */
    mtogglebutton1.setOnClickListener(new OnClickListener()
    {
      public void onClick(View v)
      {
        if (mtogglebutton1.isChecked())
        {
          /* 设置震动的周期 */
          mVibrator01.vibrate( new long[]{100,10,100,1000},-1);
          /*用 Toast 显示震动启动*/
          Toast.makeText
          (
            example10.this,
            getString(R.string.str_ok),
            Toast.LENGTH_SHORT
          ).show();
        }
        else
```

```
        {
          /* 取消震动 */
          mVibrator01.cancel();
          /*用 Toast 显示震动已被取消*/
          Toast.makeText
          (
            example10.this,
            getString(R.string.str_end),
            Toast.LENGTH_SHORT
          ).show();
        }
      }
    });
```

(3) 设置长震动模式，通过 mVibrator01.vibrate(new long[]{100,100,100,1000},0)设置了此模式下的震动周期。具体代码如下所示：

```
    /* 长震动 */
    mtogglebutton2.setOnClickListener(new OnClickListener()
    {
      public void onClick(View v)
      {
        if (mtogglebutton2.isChecked())
        {
          /*设置震动的周期*/
          mVibrator01.vibrate(new long[]{100,100,100,1000},0);

          /*用 Toast 显示震动启动*/
          Toast.makeText
          (
            example10.this,
            getString(R.string.str_ok),
            Toast.LENGTH_SHORT
          ).show();
        }
        else
        {
          /* 取消震动 */
          mVibrator01.cancel();

          /* 用 Toast 显示震动取消 */
          Toast.makeText
          (
            example10.this,
            getString(R.string.str_end),
            Toast.LENGTH_SHORT
          ).show();
        }
      }
```

```
    });
```

(4)　设置节奏震动模式，通过 mVibrator01.vibrate(new long[]{1000,50,1000,50,1000},0)设置了此模式下的震动周期。具体代码如下所示：

```
    /* 节奏震动 */
    mtogglebutton3.setOnClickListener(new OnClickListener()
    {
      public void onClick(View v)
      {
        if (mtogglebutton3.isChecked())
        {
          /* 设置震动的周期 */
          mVibrator01.vibrate( new long[]{1000,50,1000,50,1000},0);

          /*用 Toast 显示震动启动*/
          Toast.makeText
          (
            example10.this, getString(R.string.str_ok),
            Toast.LENGTH_SHORT
          ).show();
        }
        else
        {
          /* 取消震动 */
          mVibrator01.cancel();
          /* 用 Toast 显示震动取消 */
          Toast.makeText
          (
            example10.this,
            getString(R.string.str_end),
            Toast.LENGTH_SHORT
          ).show();
        }
      }
    });
  }
}
```

另外，还需要设置 AndroidManifest.xml 中的权限，即必须允许 Android.permission.VIBRATE 的权限，具体代码如下所示：

```
  <uses-permission android:name="android.permission.VIBRATE" />
```

至此整个展示结束，执行后的效果如图 10-9 所示，当选择一种模式并单击按钮后会启动对应的震动模式，如图 10-10 所示是启动了“短时间”震动模式。





图 10-9　初始效果



图 10-10　短时间震动

10.6　图 文 提 醒

题　目	目　的	源码路径
题目 5	实现图文提醒功能	“光盘:\daima\10\example5”文件夹

10.6.1　我的想法

我知道 Toast 能够在手机中实现一个醒目的提示效果，同样应该可以在 Toast 中放置一幅图片并辅助一行文字，这样即可实现图文提醒功能。图文提醒和单独的文字提醒相比，具有更好的视觉冲击效果。在具体实现上，我在 Toast 中放置了一个 Layout，这个 Layout 中包含了图片和文字，这些图片和文字就是提醒的内容。

10.6.2　具体实现

编写的主程序文件是 example5.java，具体实现流程如下。

(1)　设置 Button 用 OnClickListener 启动事件 setOnClickListener(new Button.OnClickListener()。

(2)　分别创建 ImageView 对象和 LinearLayout 对象，并用 mTextView 获取 String 值。

(3)　判断 mTextView 的内容是什么，并与系统实现连接，最后用 Toast 方式将内容显示出来。

文件 example5.java 的具体代码如下所示：

```
public class example5 extends Activity
{
```

```
private Button mButton01;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  mButton01 = (Button)findViewById(R.id.myButton1);

  /*设置 Button 用 OnClickListener 启动事件*/
  mButton01.setOnClickListener(new Button.OnClickListener()
  {

    @Override
    public void onClick(View v)
    {
      // TODO Auto-generated method stub

      /* 创建 ImageView */
      ImageView mView01 = new ImageView(example6.this);
      TextView mTextView = new TextView(example6.this);

      /* 创建 LinearLayout 对象 */
      LinearLayout lay = new LinearLayout(example6.this);

      /* 设置 mTextView 去抓取 string 值 */
      mTextView.setText(R.string.app_url);

      /* 判断 mTextView 的内容为何，并与系统做连接 */
      Linkify.addLinks
      (
        mTextView,Linkify.WEB_URLS|
        Linkify.EMAIL_ADDRESSES|
        Linkify.PHONE_NUMBERS
      );

      /*用 Toast 方式显示*/
      Toast toast = Toast.makeText
                    (
                      example6.this,
                      mTextView.getText(),
                      Toast.LENGTH_LONG
                    );

      /* 自定义 View 对象 */
      View textView = toast.getView();

      /* 以水平方式排列 */
      lay.setOrientation(LinearLayout.HORIZONTAL);

      /* 在 ImageView Widget 里指定显示的图片 */
```

```
        mView01.setImageResource(R.drawable.icon);

        /* 在 Layout 里添加刚创建的 View */
        lay.addView(mView01);

        /* 在 Toast 里显示文字 */
        lay.addView(textView);

        /* 以 Toasr,setView 方法将 Layout 传入 */
        toast.setView(lay);

        /* 显示 Toast */
        toast.show();
      }
    });
  }
}
```

至此整个展示结束，执行后的效果如图 10-11 所示，当单击“有图片的提醒”按钮后，会弹出图文效果的提醒，如图 10-12 所示。



图 10-11　执行后的效果



图 10-12　图文提醒

10.7　状态栏的亲切提醒

题　目	目　的	源码路径
题目 6	在 Android 状态栏中实现图文提醒功能	“光盘:\daima\10\example6”文件夹

10.7.1　我的想法

《Android SDK 4.0 密录》说道：在手机的最顶端，通常用于显示时间、信号强度和电池用量，这就是状态栏。在实现提醒处理时，也可以在状态栏中显示一幅图片，并结合图片和文字来实现更加美观的提示。根据秘籍提示，我决定在状态栏中放置一幅提醒图片。在具体实现上，将 Notification 添加到 NotificationManage 即可将信息显示在状态栏。

10.7.2　具体实现

编写的主程序文件是 example6.java 和 example6_1.java。

1. 文件 example6.java

(1) 声明对象变量，分别设置“在线”、“离开”、“忙碌”、“一会回来”、“离线”五种状态。具体代码如下所示：

```
public class example6 extends Activity
{
  /*声明对象变量*/
  private NotificationManager myNotiManager;
  private Spinner mySpinner;
  private ArrayAdapter<String> myAdapter;
  private static final String[] status =
  { "在线","离开","忙碌","一会回来","离线" };

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    /* 载入 main.xml Layout */
    setContentView(R.layout.main);
```

(2) 初始化 NotiManager 对象，然后使用 myspinner_dropdown 自定义下拉菜单模式，并将 ArrayAdapter 添加到 Spinner 对象中，根据具体状态显示对应的提示图标。具体代码如下所示：

```
    /* 初始化对象 */
    myNotiManager=
      (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
    mySpinner=(Spinner)findViewById(R.id.mySpinner);
    myAdapter=new ArrayAdapter<String>(this,
             android.R.layout.simple_spinner_item,status);
    /* 应用 myspinner_dropdown 自定义下拉菜单模式 */
    myAdapter.setDropDownViewResource(R.layout.myspinner_dropdown);
    /* 将 ArrayAdapter 添加到 Spinner 对象中 */
    mySpinner.setAdapter(myAdapter);

    /* 将 mySpinner 添加 OnItemSelectedListener */
    mySpinner.setOnItemSelectedListener(
      new Spinner.OnItemSelectedListener()
```

```
  {
   @Override
   public void onItemSelected(AdapterView<?> arg0,View arg1,
                        int arg2,long arg3)
   {
     /* 依照选择的 item 来判断要发哪一个 notification */
     if(status[arg2].equals("在线"))
     {
       setNotiType(R.drawable.msn,"在线");
     }
     else if(status[arg2].equals("离开"))
     {
       setNotiType(R.drawable.away,"离开");
     }
     else if(status[arg2].equals("忙碌中"))
     {
       setNotiType(R.drawable.busy,"忙碌中");
     }
     else if(status[arg2].equals("一会回来"))
     {
       setNotiType(R.drawable.min,"一会回来");
     }
     else
     {
       setNotiType(R.drawable.offine,"离线");
     }
   }
```

(3) 定义 onNothingSelected(AdapterView<?> arg0)供用户选择状态。具体代码如下所示：

```
   @Override
   public void onNothingSelected(AdapterView<?> arg0)
   {
   }
  });
}

/* 发出 Notification 的 method */
private void setNotiType(int iconId, String text)
{
  /* 创建新的 Intent，作为点击 Notification 留言条时，
   * 会运行的 Activity */
  Intent notifyIntent=new Intent(this,example7_1.class);
  notifyIntent.setFlags( Intent.FLAG_ACTIVITY_NEW_TASK);
  /* 创建 PendingIntent 作为设置递延运行的 Activity */
  PendingIntent appIntent=PendingIntent.getActivity(example7.this,
                    0,notifyIntent,0);
```

(4) 创建 Notication 并设置相关对应的参数。具体代码如下所示：

```
  Notification myNoti=new Notification();
  /* 设置 statusbar 显示的 icon */
  myNoti.icon=iconId;
```

```
    /* 设置 statusbar 显示的文字信息 */
    myNoti.tickerText=text;
    /* 设置 notification 发生时同时发出默认声音 */
    myNoti.defaults=Notification.DEFAULT_SOUND;
    /* 设置 Notification 留言条的参数 */
    myNoti.setLatestEventInfo(example7.this,"MSN 登录状态", text,appIntent);
    /* 送出 Notification */
    myNotiManager.notify(0,myNoti);
  }
}
```

2. 文件 example6_1.java

此功能是引入 example6_1 extends Activity 并发出提示，实现模拟 QQ 和 MSN 的登录效果。其主要代码如下所示：

```
/* 当 user 点击 Notification 留言条时，会运行 Activity */
public class example6_1 extends Activity
{
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);

    /* 发出 Toast */
    Toast.makeText(example6_1.this, "模拟 MSN 切换登录状态",Toast.LENGTH_SHORT
).show();
    finish();
  }
}
```

至此整个展示结束，执行后将首先显示默认的“在线”状态并在状态栏中显示 QQ 的在线图标，初始效果如图 10-13 所示。可以在下拉列表框中继续选择一种状态，状态栏也会显示对应的图标，如图 10-14 所示。例如，当选择“忙碌”状态时，会在状态栏中显示对应的提示图标如图 10-15 所示，这是一个 MSN 的图标。



图 10-13　初始效果



图 10-14　图文提醒



图 10-15　忙碌图标

10.8　模拟实现文件管理器

题　目	目　的	源码路径
题目 7	Android 实现文件管理	“光盘:\daima\10\example7”文件夹

10.8.1 我的想法

《Android SDK 4.0 密录》写道：在手机应用中，经常利用手机来存储各种各样的文件信息，这样就需要一个很好的工具来实现对各个文件的管理。可以通过 ListActivity 和 Java I/O 来查找根目录下的所有文件，并通过 setListAdapter 方法将存放文件信息的 ArrayAdapter 设置给 ListView。而 Android API 提供的 ArrayAdapter 对象只允许存入 String 数组或 List 对象，因此在显示文件列表时只能以字符串的形式来显示文件名。如果要同时显示文件名、图标和文件夹名，则需要定义一个实现 Adapter Interface 的对象。在 Android API 中提供了 BaseAdapter 对象，只要继承了此对象，就可以实现属于自己的 Adapter。在本实例中，我们事先准备了素材图片作为图标，用于代表不同的文件或文件夹类型。

10.8.2 具体实现

编写的主程序文件是 example7.java 和 MyAdapter.java。

1. 主程序文件 example7.java

(1) 声明需要的变量，其中 items 表示存放显示的名称，paths 表示存放文件路径，rootPath 表示起始目录。具体代码如下所示：

```
private List<String> items=null;
private List<String> paths=null;
private String rootPath="/";
private TextView mPath;
```

(2) 载入主布局文件 main.xml，然后初始化 mPath 用于显示当前路径。具体代码如下所示：

```
 protected void onCreate(Bundle icicle)
{
  super.onCreate(icicle);
    /* 载入main.xml Layout */
  setContentView(R.layout.main);
  /* 初始化mPath，用以显示目前路径 */
  mPath=(TextView)findViewById(R.id.mPath);
  getFileDir(rootPath);
}
```

(3) 定义方法 getFileDir(String filePath)，用于获取文件的具体架构。具体代码如下所示：

```
/* 取得文件架构的方法 */
private void getFileDir(String filePath)
{
  /* 设置目前所在路径 */
  mPath.setText(filePath);
  items=new ArrayList<String>();
  paths=new ArrayList<String>();
  File f=new File(filePath);
  File[] files=f.listFiles();
```

```
  if(!filePath.equals(rootPath))
  {
    /* 第一笔设置为[回到根目录] */
    items.add("b1");
    paths.add(rootPath);
    /* 第二笔设置为[回到上一层] */
    items.add("b2");
    paths.add(f.getParent());
  }
  /* 将所有文件添加到 ArrayList 中 */
  for(int i=0;i<files.length;i++)
  {
    File file=files[i];
    items.add(file.getName());
    paths.add(file.getPath());
  }

  /* 使用自定义的 MyAdapter 来将数据传入 ListActivity */
  setListAdapter(new MyAdapter(this,items,paths));
}
```

(4) 定义按钮触发事件处理方法 onListItemClick，如果是文件夹就再运行 getFileDir()，如果是文件就运行 openFile()。具体代码如下所示：

```
/* 设置 ListItem 被点击时要做的动作 */
@Override
protected void onListItemClick(ListView l,View v,int position,
                                long id)
{
  File file=new File(paths.get(position));
  if (file.isDirectory())
  {
    /* 如果是文件夹就再运行 getFileDir() */
    getFileDir(paths.get(position));
  }
  else
  {
    /* 如果是文件就运行 openFile() */
    openFile(file);
  }
}
```

(5) 定义方法 openFile(File f)，用于在手机上打开指定的文件。具体代码如下所示：

```
/* 在手机上打开文件的方法 */
private void openFile(File f)
{
  Intent intent = new Intent();
  intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
  intent.setAction(android.content.Intent.ACTION_VIEW);

  /* 调用 getMimeType()来取得 MimeType */
  String type = getMimeType(f);
```

```
    /* 设置 intent 的 file 与 MimeType */
    intent.setDataAndType(Uri.fromFile(f),type);
    startActivity(intent);
  }
```

(6) 定义 getMimeType(File f)方法，用于判断文件的类型，其中“end”是结尾的扩展名，只需获取“end”即可。具体代码如下所示：

```
  /* 判断文件 MimeType 的方法 */
  private String getMimeType(File f)
  {
    String type="";
    String fName=f.getName();
    /* 取得扩展名 */
    String end=fName.substring(fName.lastIndexOf(".")+1,
                 fName.length()).toLowerCase();

    /* 依附档名的类型决定 MimeType */
    if(end.equals("m4a")||end.equals("mp3")||end.equals("mid")
       ||end.equals("xmf")||end.equals("ogg")||end.equals("wav"))
    {
      type = "audio";
    }
    else if(end.equals("3gp")||end.equals("mp4"))
    {
      type = "video";
    }
    else if(end.equals("jpg")||end.equals("gif")||end.equals("png")
           ||end.equals("jpeg")||end.equals("bmp"))
    {
      type = "image";
    }
    else
    {
      /* 如果无法直接打开，就跳出软件列表给用户选择 */
      type="*";
    }
    type += "/*";
    return type;
  }
}
```

2. 文件 MyAdapter.java

(1) 定义 MyAdapter 构造器并分别传入 items、paths 和 mInflater 这三个参数，最后对前面定义的四个变量进行赋值。具体代码如下所示：

```
  /* MyAdapter 的构造器，传入三个参数  */
  public MyAdapter(Context context,List<String> it,List<String> pa)
  {
    /* 参数初始化 */
    mInflater = LayoutInflater.from(context);
    items = it;
```

```
    paths = pa;
    mIcon1 = BitmapFactory.decodeResource(context.getResources(),
                                   R.drawable.back01);
    mIcon2 = BitmapFactory.decodeResource(context.getResources(),
                                   R.drawable.back02);
    mIcon3 = BitmapFactory.decodeResource(context.getResources(),
                                   R.drawable.folder);
    mIcon4 = BitmapFactory.decodeResource(context.getResources(),
                                   R.drawable.doc);
  }
```

(2) 分别覆盖 getCount()、getItem(int position)、getItemId(int position)。具体代码如下所示：

```
/* 因继承 BaseAdapter，需覆盖以下方法 */
@Override
public int getCount()
{
  return items.size();
}

@Override
public Object getItem(int position)
{
  return items.get(position);
}

@Override
public long getItemId(int position)
{
  return position;
}
```

(3) 使用自定义的 file_row 作为 Layout，然后分别设置“回到根目录”的文字与 icon 和“回到上一层”的文字与 icon。具体代码如下所示：

```
@Override
public View getView(int position,View convertView,ViewGroup parent)
{
  ViewHolder holder;

  if(convertView == null)
  {
    /* 使用自定义的 file_row 作为 Layout */
    convertView = mInflater.inflate(R.layout.file_row, null);
    /* 初始化 holder 的 text 与 icon */
    holder = new ViewHolder();
    holder.text = (TextView) convertView.findViewById(R.id.text);
    holder.icon = (ImageView) convertView.findViewById(R.id.icon);

    convertView.setTag(holder);
  }
  else
  {
    holder = (ViewHolder) convertView.getTag();
```

```
    }

    File f=new File(paths.get(position).toString());
    /* 设置[回到根目录]的文字与 icon */
    if(items.get(position).toString().equals("b1"))
    {
      holder.text.setText("Back to /");
      holder.icon.setImageBitmap(mIcon1);
    }
    /* 设置[回到上一层]的文字与 icon */
    else if(items.get(position).toString().equals("b2"))
    {
      holder.text.setText("Back to ..");
      holder.icon.setImageBitmap(mIcon2);
    }
    /* 设置[文件或文件夹]的文字与 icon */
    else
    {
      holder.text.setText(f.getName());
      if(f.isDirectory())
      {
        holder.icon.setImageBitmap(mIcon3);
      }
      else
      {
        holder.icon.setImageBitmap(mIcon4);
      }
    }
    return convertView;
  }
  /* class ViewHolder */
  private class ViewHolder
  {
    TextView text;
    ImageView icon;
  }
}
```

至此整个演示结束，执行后的初始效果如图 10-16 所示。当选择一个文件夹后会继续显示此文件夹里面的内容，文件信息如图 10-17 所示。

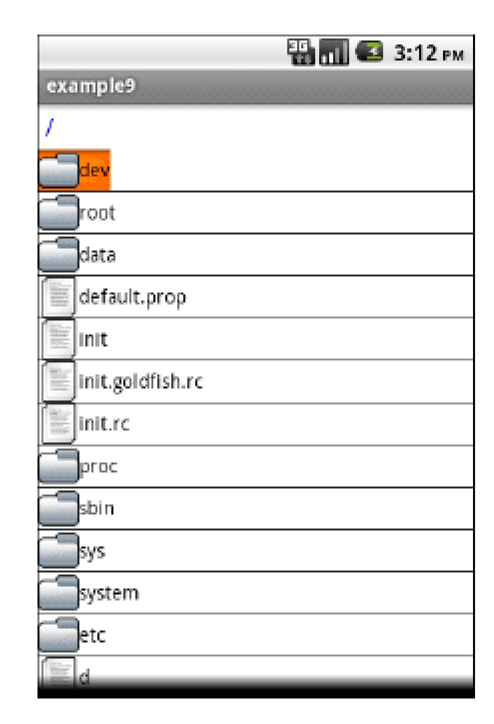


图 10-16 初始效果

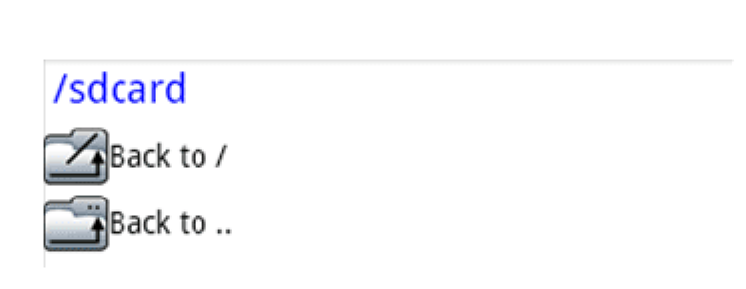


图 10-17 文件信息

10.9　控制 WiFi 服务

题　目	目　的	源码路径
题目 8	在 Android 系统中对 WiFi 进行控制	“光盘:\daima\10\example8”文件夹

10.9.1　我的想法

秘籍中写道：WiFi 是一种可以将个人电脑、手持设备(如 PDA、手机)等终端以无线方式互相连接的技术。WiFi 是一个无线网路通信技术的品牌，由 WiFi 联盟(WiFi Alliance)所持有。目的是改善基于 IEEE 802.11 标准的无线网路产品之间的互通性。通常用户在 Android 系统中，存在了一个无线控制模块，打开方式为依次单击 Menu | Settings | Wireless$networks | WiFi settings，菜单命令，出现图 10-18 所示的界面。

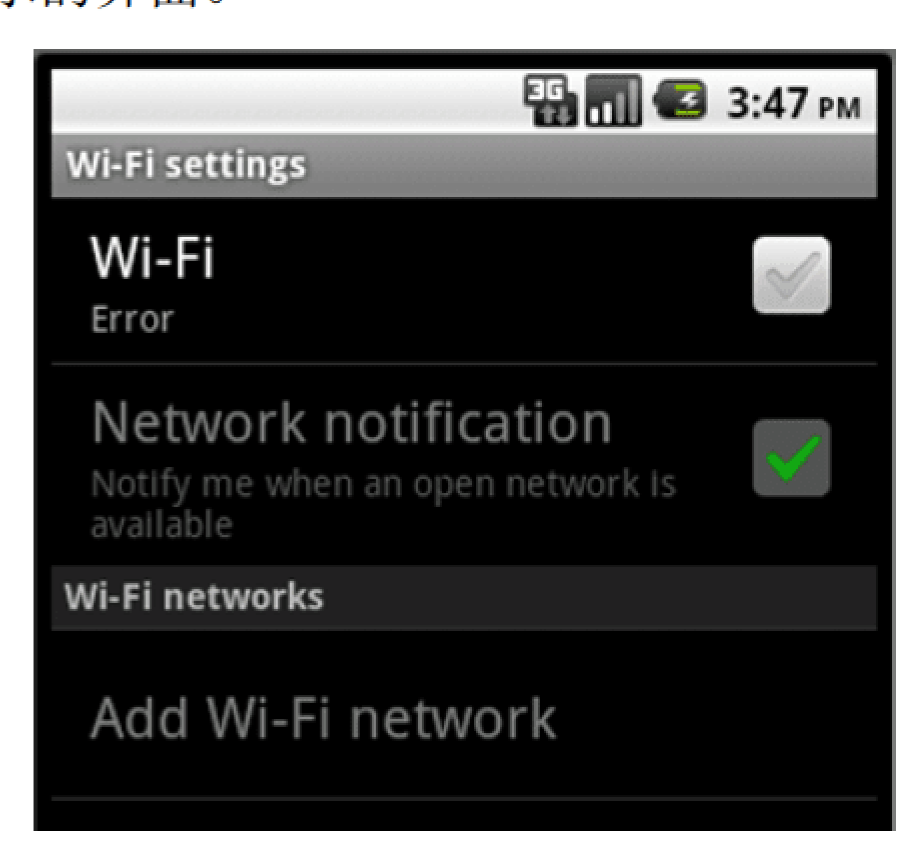


图 10-18　WiFi 控制界面

图 10-18 所示的是 WiFi 控制界面，在此可以控制 WiFi 的打开和关闭，此时我知道了怎样以编程的方式实现同样类似的功能，我准备用 WifiManager 提供的如下 8 种状态。

- WifiManager.WIFI_STATE_ENABLING：表示 WiFi 已经打开。
- WifiManager.WIFI_STATE_DISABLING：表示 WiFi 正在关闭而无法关闭。
- WifiManager.WIFI_STATE_DISABLED：表示 WiFi 已经关闭。
- WifiManager.WIFI_STATE_ENABLING：表示 WiFi 正在打开而无法关闭。
- WifiManager.WIFI_STATE_ENABLED：表示 WiFi 已经打开无法再打开。
- WifiManager.WIFI_STATE_DISABLING：表示 WiFi 正在关闭。
- WifiManager.WIFI_STATE_DISABLED：表示 WiFi 已经关闭。
- WifiManager.WIFI_STATE_UNKNOWN：表示 WiFi 无法识别。

在具体实现上，先定义一个复选框 CheckBox，然后捕捉 CheckBox 的点击事件，根据对应的状态显示对应的提示。例如，可以用下面的代码检测 WiFi 是否启动：

```
WifiManager wm = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
if(wm.getWifiState() == WifiManager.WIFI_STATE_ENABLED){
   return true;
}
```

10.9.2　具体实现

编写的主程序文件是 example8.java，具体实现流程如下。

(1)　创建 WiFiManager 对象 mWiFiManager01。具体代码如下所示：

```
public class example8 extends Activity
{
  private TextView mTextView01;
  private CheckBox mCheckBox01;

  /* 创建 WiFiManager 对象 */
  private WifiManager mWiFiManager01;
```

(2)　定义 mTextView01 和 mCheckBox01，分别用于显示提示文本和获取复选框的选择状态。具体代码如下所示：

```
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mTextView01 = (TextView) findViewById(R.id.myTextView1);
    mCheckBox01 = (CheckBox) findViewById(R.id.myCheckBox1);
```

(3)　以 getSystemService 取得 WIFI_SERVICE。具体代码如下所示：

```
    mWiFiManager01 = (WifiManager)
      this.getSystemService(Context.WIFI_SERVICE);
```

(4)　通过 if 语句来判断运行程序后的 WiFi 状态是否打开或打开中，这样便可以显示对应的提示信息。具体代码如下所示：

```
    /* 判断运行程序后的 WiFi 状态是否打开或打开中 */
    if(mWiFiManager01.isWifiEnabled())
    {
      /* 判断 WiFi 状态是否“已打开” */
      if(mWiFiManager01.getWifiState()==
         WifiManager.WIFI_STATE_ENABLED)
      {
        /* 若 WiFi 已打开，将选取项打勾 */
        mCheckBox01.setChecked(true);
        /* 更改选取项文字为关闭 WiFi*/
        mCheckBox01.setText(R.string.str uncheck);
      }
      else
      {
        /* 若 WiFi 未打开，将选取项勾选择取消 */
        mCheckBox01.setChecked(false);
        /* 更改选取项文字为打开 WiFi*/
        mCheckBox01.setText(R.string.str_checked);
```

```
    }
  }
  else
  {
    mCheckBox01.setChecked(false);
    mCheckBox01.setText(R.string.str_checked);
  }
```

(5) 通过 mCheckBox01.setOnClickListener 来捕捉 CheckBox 的点击事件，用 onClick(View v)方法获取用户的点击，然后根据 if 语句的操作需求来执行对应操作，并根据需要输出对应的提示信息。具体代码如下所示：

```
mCheckBox01.setOnClickListener(
new CheckBox.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    // TODO Auto-generated method stub

    /* 当选取项为取消选取状态 */
    if(mCheckBox01.isChecked()==false)
    {
      /* 尝试关闭 Wi-Fi 服务 */
      try
      {
        /* 判断 WiFi 状态是否为已打开 */
        if(mWiFiManager01.isWifiEnabled() )
        {
          /* 关闭 WiFi  */
          if(mWiFiManager01.setWifiEnabled(false))
          {
            mTextView01.setText(R.string.str_stop_wifi_done);
          }
          else
          {
            mTextView01.setText(R.string.str_stop_wifi_failed);
          }
        }
        else
        {
          /* WiFi 状态不为已打开状态时 */
          switch(mWiFiManager01.getWifiState())
          {
            /* WiFi 正在打开过程中，导致无法关闭... */
            case WifiManager.WIFI_STATE_ENABLING:
              mTextView01.setText
              (
                getResources().getText
                (R.string.str_stop_wifi_failed)+":"+
                getResources().getText
                (R.string.str_wifi_enabling)
```

```
            );
            break;
          /* WiFi 正在关闭过程中，导致无法关闭... */
          case WifiManager.WIFI_STATE_DISABLING:
            mTextView01.setText
            (
              getResources().getText
              (R.string.str_stop_wifi_failed)+":"+
              getResources().getText
              (R.string.str_wifi_disabling)
            );
            break;
          /* WiFi 已经关闭 */
          case WifiManager.WIFI_STATE_DISABLED:
            mTextView01.setText
            (
              getResources().getText
              (R.string.str_stop_wifi_failed)+":"+
              getResources().getText
              (R.string.str_wifi_disabled)
            );
            break;
          /* 无法取得或辨识 WiFi 状态 */
          case WifiManager.WIFI_STATE_UNKNOWN:
          default:
            mTextView01.setText
            (
              getResources().getText
              (R.string.str_stop_wifi_failed)+":"+
              getResources().getText
              (R.string.str_wifi_unknow)
            );
            break;
        }
        mCheckBox01.setText(R.string.str_checked);
      }
    }
    catch (Exception e)
    {
      Log.i("HIPPO", e.toString());
      e.printStackTrace();
    }
  }
  else if(mCheckBox01.isChecked()==true)
  {
    /* 尝试打开 WiFi 服务 */
    try
    {
      /* 确认 WiFi 服务是关闭且不在打开作业中 */
      if(!mWiFiManager01.isWifiEnabled() &&
         mWiFiManager01.getWifiState()!=
         WifiManager.WIFI_STATE_ENABLING )
```

```
{
  if(mWiFiManager01.setWifiEnabled(true))
  {
    switch(mWiFiManager01.getWifiState())
    {
      /* WiFi 正在打开过程中，导致无法打开... */
      case WifiManager.WIFI_STATE_ENABLING:
        mTextView01.setText
        (
          getResources().getText
          (R.string.str_wifi_enabling)
        );
        break;
      /* WiFi 已经为打开，无法再次打开... */
      case WifiManager.WIFI_STATE_ENABLED:
        mTextView01.setText
        (
          getResources().getText
          (R.string.str_start_wifi_done)
        );
        break;
      /* 其他未知的错误 */
      default:
        mTextView01.setText
        (
          getResources().getText
          (R.string.str_start_wifi_failed)+":"+
          getResources().getText
          (R.string.str_wifi_unknow)
        );
        break;
    }
  }
  else
  {
    mTextView01.setText(R.string.str_start_wifi_failed);
  }
}
else
{
  switch(mWiFiManager01.getWifiState())
  {
    /* WiFi 正在打开过程中，导致无法打开... */
    case WifiManager.WIFI_STATE_ENABLING:
      mTextView01.setText
      (
        getResources().getText
        (R.string.str_start_wifi_failed)+":"+
        getResources().getText
        (R.string.str_wifi_enabling)
      );
      break;
```

```
                /* WiFi 正在关闭过程中，导致无法打开... */
                case WifiManager.WIFI_STATE_DISABLING:
                  mTextView01.setText
                  (
                    getResources().getText
                    (R.string.str_start_wifi_failed)+":"+
                    getResources().getText
                    (R.string.str_wifi_disabling)
                  );
                  break;
                /* WiFi 已经关闭 */
                case WifiManager.WIFI STATE DISABLED:
                  mTextView01.setText
                  (
                    getResources().getText
                    (R.string.str_start_wifi_failed)+":"+
                    getResources().getText
                    (R.string.str_wifi_disabled)
                  );
                  break;
                /* 无法取得或识别 WiFi 状态 */
                case WifiManager.WIFI_STATE_UNKNOWN:
                default:
                  mTextView01.setText
                  (
                    getResources().getText
                    (R.string.str_start_wifi_failed)+":"+
                    getResources().getText
                    (R.string.str_wifi_unknow)
                  );
                  break;
              }
            }
            mCheckBox01.setText(R.string.str_uncheck);
          }
          catch (Exception e)
          {
            Log.i("HIPPO", e.toString());
            e.printStackTrace();
          }
        }
      }
    });
  }
```

(6) 定义 mMakeTextToast(String str, boolean isLong)，用于根据当前操作显示对应的提示性信息。具体代码如下所示：

```
public void mMakeTextToast(String str, boolean isLong)
{
  if(isLong==true)
  {
```

```
      Toast.makeText(example110.this, str, Toast.LENGTH LONG).show();
    }
    else
    {
      Toast.makeText(example110.this, str, Toast.LENGTH_SHORT).show();
    }
  }

  @Override
  protected void onResume()
  {
    // TODO Auto-generated method stub

    /* 在 onResume 重写事件为取得打开程序当下 WiFi 的状态 */
    try
    {
      switch(mWiFiManager01.getWifiState())
      {
        /* WiFi 已经在打开状态... */
        case WifiManager.WIFI_STATE_ENABLED:
          mTextView01.setText
          (
            getResources().getText(R.string.str_wifi_enabling)
          );
          break;
        /* WiFi 正在打开过程中状态... */
        case WifiManager.WIFI_STATE_ENABLING:
          mTextView01.setText
          (
            getResources().getText(R.string.str_wifi_enabling)
          );
          break;
        /* WiFi 正在关闭过程中... */
        case WifiManager.WIFI_STATE_DISABLING:
          mTextView01.setText
          (
            getResources().getText(R.string.str wifi disabling)
          );
          break;
        /* WiFi 已经关闭 */
        case WifiManager.WIFI_STATE_DISABLED:
          mTextView01.setText
          (
            getResources().getText(R.string.str_wifi_disabled)
          );
          break;
        /* 无法取得或识别 WiFi 状态 */
        case WifiManager.WIFI_STATE_UNKNOWN:
        default:
          mTextView01.setText
          (
            getResources().getText(R.string.str_wifi_unknow)
```

```
        );
        break;
      }
    }
    catch(Exception e)
    {
      mTextView01.setText(e.toString());
      e.getStackTrace();
    }
    super.onResume();
  }

  @Override
  protected void onPause()
  {
    // TODO Auto-generated method stub
    super.onPause();
  }
}
```

接下来，需要在文件 AndroidManifest.xml 中添加对 WiFi 的访问以及网络状态的权限。具体代码如下所示：

```
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

至此整个展示结束，执行后会显示一个按钮，如图 10-19 所示。当选择复选框后会执行对应的操作处理并且显示对应的提示信息，如图 10-20 所示。

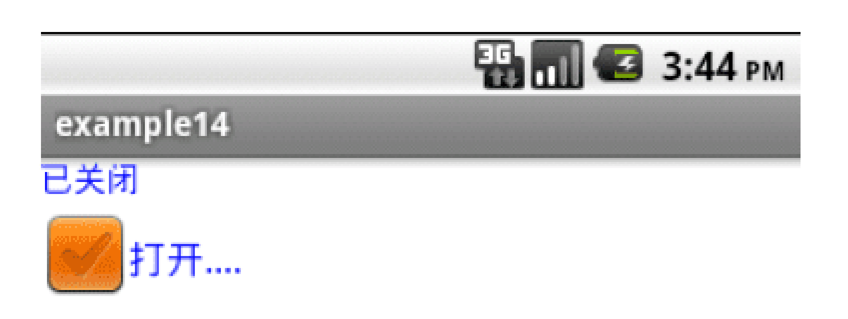


图 10-19　执行后的效果



图 10-20　提示信息

10.10　获取 SIM 卡内信息

题　目	目　的	源码路径
题目 9	在 Android 系统中获取 SIM 卡内信息	“光盘:\daima\10\example9”文件夹

10.10.1　何谓 SIM 卡

翻开《Android SDK 4.0 密录》见到上面写道：SIM 卡是(Subscriber Identity Module 客户识



别模块)的缩写，也称为智能卡、用户身份识别卡，GSM 数字移动电话机必须装上此卡才能使用。它在一电脑芯片上存储了数字移动电话客户的信息，加密的密钥等内容，可供 GSM 网络客户进行身份鉴别，并对客户通话时的语音信息进行加密。SIM 卡的使用，完全防止了并机和通话被窃听的行为，并且 SIM 卡的制作是严格按照 GSM 国际标准和规范来完成的，从而可靠的保障了客户的正常通信。为防止他人擅用您的 SIM 卡，SIM 卡设置了个人识别密码 PIN 码，只要设置了 PIN 码用户在每次打开手机时，屏幕上会显示要求输入 4 位 PIN 码。

10.10.2　我的想法

了解了何谓 SIM 卡之后，整个题目的实现过程就很明确了。我可以用 Android API 中的 TelephonyManager 对象的方法实现对 SIM 卡信息的读取。另外，TelephonyManager 还能获取手机的号码，还提供了多种获取手机信息的函数，例如 imei、imsi 等。

10.10.3　具体实现

本实例的主程序文件是 example9.java 和 MyAdapter.java，下面开始讲解其具体的实现代码。

1. 文件 example9.java

(1)　先载入 main.xml Layout，然后通过 add(getResources().getText(R.string.str_list0).toString()) 将取得的信息写入 List 中，最后通过 if 语句来设置 SIM 卡的状态。具体代码如下所示：

```
public class example9 extends ListActivity
{
  private TelephonyManager telMgr;
  private List<String> item=new ArrayList<String>();
  private List<String> value=new ArrayList<String>();

  @SuppressWarnings("static-access")
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    /* 载入 main.xml Layout */
    setContentView(R.layout.main);
    telMgr = (TelephonyManager)getSystemService(TELEPHONY_SERVICE);

    /* 将取得的信息写入 List 中 */
    /* 取得 SIM 卡状态 */
    item.add(getResources().getText(R.string.str_list0).toString());
    if(telMgr.getSimState()==telMgr.SIM_STATE_READY)
    {
      value.add("良好");
    }
    else if(telMgr.getSimState()==telMgr.SIM_STATE_ABSENT)
    {
      value.add("无 SIM 卡");
```

```
}
else
{
  value.add("SIM 卡被锁定或未知的状态");
}
```

(2) 分别获取 SIM 卡的卡号、SIM 卡供货商代码、SIM 卡供货商名称、SIM 卡国别，然后使用自定义的 MyAdapter 将数据传入 ListActivity。具体代码如下所示：

```
item.add(getResources().getText(R.string.str_list1).toString());
if(telMgr.getSimSerialNumber()!=null)
{
  value.add(telMgr.getSimSerialNumber());
}
else
{
  value.add("无法取得");
}

/* 取得 SIM 卡供货商代码 */
item.add(getResources().getText(R.string.str_list2).toString());
if(telMgr.getSimOperator().equals(""))
{
  value.add("无法取得");
}
else
{
  value.add(telMgr.getSimOperator());
}

/* 取得 SIM 卡供货商名称 */
item.add(getResources().getText(R.string.str_list3).toString());
if(telMgr.getSimOperatorName().equals(""))
{
  value.add("无法取得");
}
else
{
  value.add(telMgr.getSimOperatorName());
}

/* 取得 SIM 卡国别 */
item.add(getResources().getText(R.string.str list4).toString());
if(telMgr.getSimCountryIso().equals(""))
{
  value.add("无法取得");
}
else
{
  value.add(telMgr.getSimCountryIso());
}
```

```
    /* 使用自定义的MyAdapter来将数据传入ListActivity */
    setListAdapter(new MyAdapter(this,item,value));
  }
}
```

2. 文件 MyAdapter.java

(1) 声明三个变量 mInflater、items 和 values，然后定义 MyAdapter 构造器传入三个参数，并实现参数初始化。具体代码如下所示：

```
/* 自定义的Adapter，继承android.widget.BaseAdapter */
public class MyAdapter extends BaseAdapter
{
  /* 变量声明 */
  private LayoutInflater mInflater;
  private List<String> items;
  private List<String> values;
  /* MyAdapter的构造器，传入三个参数  */
  public MyAdapter(Context context,List<String> item,
                List<String> value)
  {
    /* 参数初始化 */
    mInflater = LayoutInflater.from(context);
    items = item;
    values = value;
  }
```

(2) 分别覆盖方法 getCount()、getItem(int position)、getItemId(int position)、getView(int position,View convertView,ViewGroup par)。具体代码如下所示：

```
  /* 因继承BaseAdapter，需覆盖以下方法 */
  @Override
  public int getCount()
  {
    return items.size();
  }

  @Override
  public Object getItem(int position)
  {
    return items.get(position);
  }

  @Override
  public long getItemId(int position)
  {
    return position;
  }

  @Override
  public View getView(int position,View convertView,ViewGroup par)
  {
    ViewHolder holder;
```

```
    if(convertView == null)
    {
      /* 使用自定义的 file_row 作为 Layout */
      convertView = mInflater.inflate(R.layout.row_layout,null);
      /* 初始化 holder 的 text 与 icon */
      holder = new ViewHolder();
      holder.text1=(TextView)convertView.findViewById(R.id.myText1);
      holder.text2=(TextView)convertView.findViewById(R.id.myText2);

      convertView.setTag(holder);
    }
    else
    {
      holder = (ViewHolder) convertView.getTag();
    }
    /* 设置要显示的信息 */
    holder.text1.setText(items.get(position).toString());
    holder.text2.setText(values.get(position).toString());

    return convertView;
  }

  /* class ViewHolder */
  private class ViewHolder
  {
    /* text1：信息名称
     * text2：信息内容 */
    TextView text1;
    TextView text2;
  }
}
```

在文件 AndroidManifest.xml 中设置读取电话状态的权限。具体代码如下所示：

```
<uses-permission android:name="android.permission.READ_PHONE_STATE">
```

至此整个展示结束，执行后会显示对应的获取信息，如图 10-21 所示。



图 10-21　执行效果

10.11　实现触摸拨号按钮

题　目	目　的	源码路径
题目 10	用 Android 实现类似触摸拨号按钮	“光盘:\daima\10\example10”文件夹

10.11.1　我的想法

我知道当前触摸手机很盛行，苹果派的主打卖点就是全屏触摸。其实在安卓中，也可以通过 Intent 方式将电话号码传递给内置的拨号程序，然后内置拨号程序实现拨号处理操作。利用 startActivity()方法将程序焦点交给内置的拨号程序，这样原来的 Activity 会成为失焦状态，并且还会发生 onPause()事件，直到关闭拨号程序，焦点也交还给原来的 Activity。在具体实现上，只需插入一个按钮，当单击按钮后会调用手机内置的默认拨号界面。

10.11.2　具体实现

编写的主程序文件是 example10.java，功能是当用户单击拨号按钮后通过 android.intent.action.CALL_BUTTON 调用默认的拨号界面。具体代码如下所示：

```
public class example10 extends Activity
{
  private ImageButton myImageButton;
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    myImageButton = (ImageButton) findViewById(R.id.myImageButton);
    myImageButton.setOnClickListener(new ImageButton.OnClickListener()
    {
      @Override
      public void onClick(View v)
      {
        /* 调用拨号的画面 */
        Intent myIntentDial = new Intent("android.intent.action.CALL_BUTTON");
        startActivity(myIntentDial);
      }
    });
  }
}
```

至此整个展示结束，执行后会显示对应的按钮，如图 10-22 所示。单击拨号按钮后，会自动来到系统内置的默认拨号界面，如图 10-23 所示。

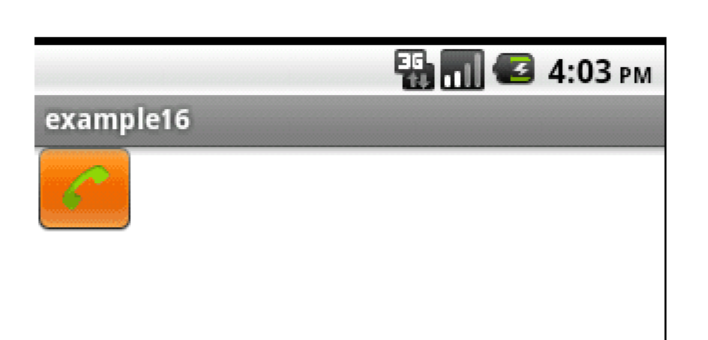



图 10-22　初始效果

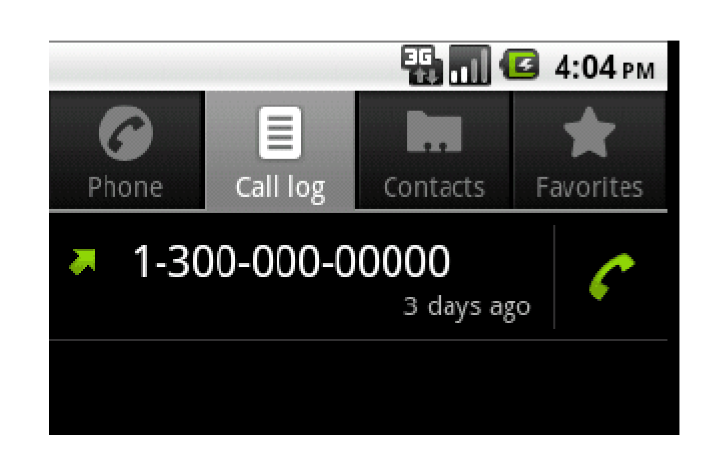


图 10-23　内置的拨号界面

10.12　查看正在运行的程序

题　目	目　的	源码路径
题目 11	查看并显示手机中当前正在运行的程序	“光盘:\daima\10\example11”文件夹

10.12.1　我的想法

我知道电脑中的题目管理器，它里面的进程管理器能够显示当前正在运行的程序。其实在手机中实现类似的功能十分简单，安卓中可以用 ActivityManager.getRunningTasks 方法来获取。可以先插入一个按钮，当单击按钮后会通过 ListView 将获取的信息显示出来。单击按钮后，如果在 ListView 的工作已经结束或被操作系统回收，则不会更新运行列表。另外，如果不具有访问其他运行程序的权限，也不会显示在 ListView 列表中。

10.12.2　具体实现

编写的主程序文件是 example11.java，具体的实现流程如下：

(1)　设置类成员最多能够获取 30 个 Task 数量，设置类成员 ActivityManager 的对象，然后定义当点击按钮后取得正在后台运行的工作程序。具体代码如下所示：

```
/* 类成员设置取回最多几笔的 Task 数量 */
private int intGetTastCounter=30;
/* 类成员 ActivityManager 对象 */
private ActivityManager mActivityManager;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  mButton01 = (Button)findViewById(R.id.myButton1);
  mListView01 = (ListView)findViewById(R.id.myListView1);

  /* 单击按钮取得正在后台运行的工作程序 */
  mButton01.setOnClickListener(new Button.OnClickListener()
```

```
{
  @Override
  public void onClick(View v)
  {
    // TODO Auto-generated method stub
    try
    {
      /* ActivityManager 对象向系统取得 ACTIVITY_SERVICE */
      mActivityManager = (ActivityManager)
      example17.this.getSystemService(ACTIVITY_SERVICE);

      arylistTask = new ArrayList<String>();

      /* 以 getRunningTasks 方法取回正在运行中的程序 TaskInfo */
      List<ActivityManager.RunningTaskInfo> mRunningTasks =
      mActivityManager.getRunningTasks(intGetTastCounter);

      int i = 1;
      /* 以循环及 baseActivity 方式取得工作名称与 ID */
      for (ActivityManager.RunningTaskInfo amTask : mRunningTasks)
      {
        /* baseActivity.getClassName 取出运行工作名称 */
        arylistTask.add("" + (i++) + ": "+
        amTask.baseActivity.getClassName()+
        "(ID=" + amTask.id +")");
      }
      aryAdapter1 = new ArrayAdapter<String>
      (example17.this, R.layout.simple_list_item_1, arylistTask);

      if(aryAdapter1.getCount()==0)
      {
        /* 当没有任何运行的工作，则提示信息 */
        mMakeTextToast
        (
          getResources().getText
          (R.string.str_err_no_running_task).toString(),
          true
        );
      }
      else
      {
        /* 发现后台运行的工作程序，以 ListView Widget 条列呈现 */
        mListView01.setAdapter(aryAdapter1);
      }
    }
    catch(SecurityException e)
    {
      /* 当无 GET_TASKS 权限时(SecurityException 异常)提示信息 */
      mMakeTextToast
      (
        getResources().getText
        (R.string.str_err_permission).toString(),
```

```
        true
      );
    }
  }
});
```

(2) 设置当用户在运行工作选择时的事件处理。具体代码如下所示：

```
mListView01.setOnItemSelectedListener
(new ListView.OnItemSelectedListener()
{
  @Override
  public void onItemSelected
  (AdapterView<?> parent, View v, int id, long arg3)
  {
    // TODO Auto-generated method stub
    /* 由于将运行工作以数组存放，所以使用 id 取出数组元素名称 */
    mMakeTextToast(arylistTask.get(id).toString(),false);
  }
  @Override
  public void onNothingSelected(AdapterView<?> arg0)
  {
    // TODO Auto-generated method stub

  }
});
```

(3) 设置当用户在运行工作选择时的事件处理。具体代码如下所示：

```
  /* 当 User 在运行工作上单击时的事件处理 */
  mListView01.setOnItemClickListener
  (new ListView.OnItemClickListener()
  {
    @Override
    public void onItemClick
    (AdapterView<?> parent, View v, int id,  long arg3)
    {
      // TODO Auto-generated method stub
      /* 由于将运行工作以数组存放，故以 id 取出数组元素名称 */
      mMakeTextToast(arylistTask.get(id).toString(), false);
    }
  });
}
```

(4) 定义 mMakeTextToast(String str, boolean isLong)方法，用于实现一个提醒。具体代码如下所示：

```
public void mMakeTextToast(String str, boolean isLong)
{
  if(isLong==true)
  {
    Toast.makeText(example17.this, str, Toast.LENGTH_LONG).show();
  }
  else
```

```
      {
        Toast.makeText(example17.this, str, Toast.LENGTH_SHORT).show();
      }
    }
}
```

最后还需要在文件 AndroidManifest.xml 中设置权限。具体代码如下所示：

```
<uses-permission
android:name="android.permission.GET_TASKS"></uses-permission>
```

至此整个展示结束，执行后会显示对应的按钮，如图 10-24 所示。单击“正在获取运行的程序”按钮后，会显示当前正在运行的程序，如图 10-25 所示。

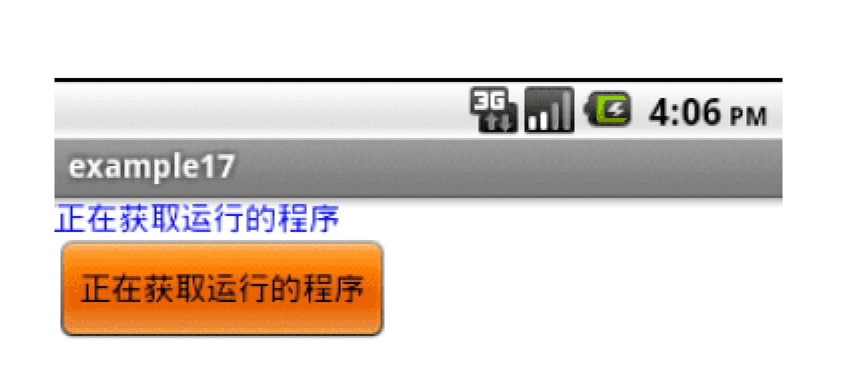


图 10-24　初始效果

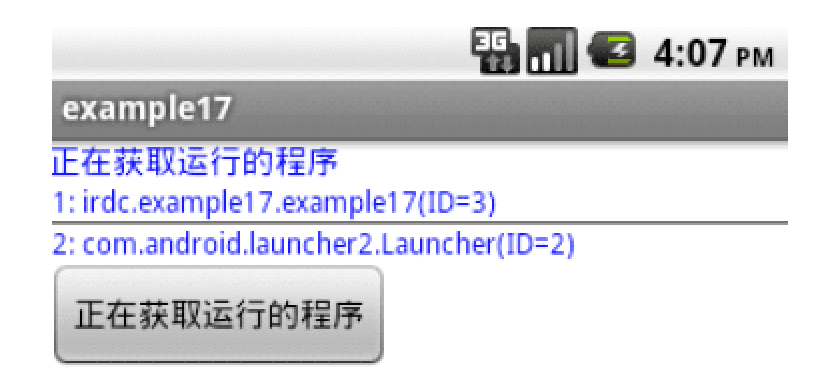


图 10-25　当前运行的程序

注意：为了保证 Android 的运行，限制了获取程序的数量，在本实例中设置了最多获取 30 个进程。

10.13　屏幕方向可以改变

题　目	目　的	源码路径
题目 12	更改手机屏幕的方向	“光盘:\daima\10\example12”文件夹

10.13.1　我的想法

屏幕旋转对我来说不会陌生，很多智能手机都能根据用户拿手机的方式，动态的横向或纵向显示屏幕中的内容。编程实现屏幕旋转很简单，先插入一个按钮，当单击按钮后会先判断当前屏幕的方向，即如果是横向显示则改为纵向显示，如果是纵向显示则改为横向显示。在具体实现上，如果要改变屏幕方向，就必须要覆盖 setRequestedOrientation()方法。如果要获取当前的屏幕方向，就需要访问 getRequestedOrientation()方法。

10.13.2　具体实现

编写的主程序文件是 example12.java，具体实现流程如下所示。

(1)　在 onCreate 方法中判断 getRequestedOrientation()是否为-1，如果是-1 则表示在 Activity 属性中没有设置 Android:screenOrientation 的值，即表示即使单击了按钮，也无法判断出屏幕的方向，不会实现屏幕方向的更改。具体代码如下所示：

```
public class example12 extends Activity
{
  private TextView mTextView01;
  private Button mButton01;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mButton01 = (Button)findViewById(R.id.myButton1);
    mTextView01 = (TextView)findViewById(R.id.myTextView1);

    if(getRequestedOrientation()==-1)
    {
      mTextView01.setText(getResources().getText
      (R.string.str err 1001));
    }
```

(2) 定义 setOnClickListener(new Button.OnClickListener())方法，当用户单击按钮后开始旋转屏幕画面。具体代码如下所示：

```
    mButton01.setOnClickListener(new Button.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
        /* 方法一：重写 getRequestedOrientation */

        /* 若无法取得 screenOrientation 属性 */
        if(getRequestedOrientation()==-1)
        {
          /* 提示无法进行画面旋转功能，因无法判别 Orientation */
          mTextView01.setText(getResources().getText
          (R.string.str err 1001));
        }
        else
        {
          if(getRequestedOrientation()==
             ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE)
          {
            /* 若当下为横排，则更改为竖排呈现 */
            setRequestedOrientation
            (ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
          }
          else if(getRequestedOrientation()==
                  ActivityInfo.SCREEN ORIENTATION PORTRAIT)
          {
            /* 若当下为竖排，则更改为横排呈现 */
            setRequestedOrientation
```

```
        (ActivityInfo.SCREEN ORIENTATION LANDSCAPE);
      }
    }
  }
 });
}
```

(3)　定义 setRequestedOrientation(int requestedOrientation)方法，判断当前要更改的屏幕方向实现旋转。具体代码如下所示：

```
@Override
public void setRequestedOrientation(int requestedOrientation)
{
  // TODO Auto-generated method stub

  /* 判断要更改的方向，以 Toast 提示 */
  switch(requestedOrientation)
  {
    /* 更改为 LANDSCAPE */
    case (ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE):
      mMakeTextToast
      (
        getResources().getText(R.string.str_msg1).toString(),
        false
      );
      break;
    /* 更改为 PORTRAIT */
    case (ActivityInfo.SCREEN_ORIENTATION_PORTRAIT):
      mMakeTextToast
      (
        getResources().getText(R.string.str msg2).toString(),
        false
      );
      break;
  }
  super.setRequestedOrientation(requestedOrientation);
}
```

(4)　定义 getRequestedOrientation(int requestedOrientation)方法，获取当前屏幕的方向。具体代码如下所示：

```
@Override
public int getRequestedOrientation()
{
  // TODO Auto-generated method stub

  /* 此重写 getRequestedOrientation 方法，可取得当下屏幕的方向 */
  return super.getRequestedOrientation();
}

public void mMakeTextToast(String str, boolean isLong)
{
  if(isLong==true)
```

```
    {
      Toast.makeText(example18.this, str, Toast.LENGTH_LONG).show();
    }
    else
    {
      Toast.makeText(example18.this, str, Toast.LENGTH_SHORT).show();
    }
  }
}
```

在 AndroidManifest.xml 中设置 Activity 的 Android:screenOrientation 属性，具体代码如下所示：

```
    <activity android:name=".example18" android:label="@string/app_name"
      android:screenOrientation="portrait"
>
```

至此整个展示结束，执行后会显示对应的按钮，如图 10-26 所示。单击“旋转处理”按钮后会实现屏幕的旋转，如图 10-27 所示。

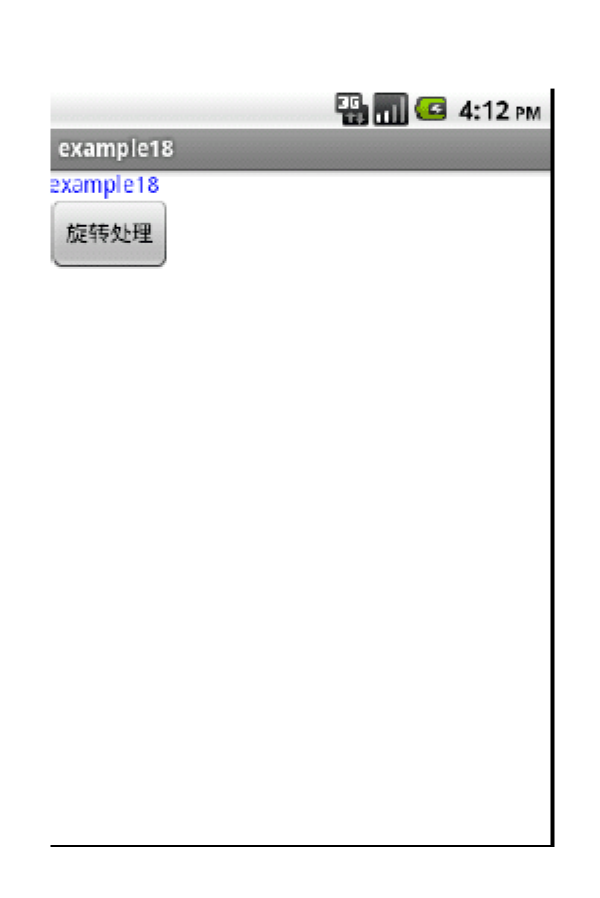


图 10-26　初始效果



图 10-27　实现旋转



第 11 章　第二关：消息埋伏的自动化

在手机应用中有很多的自动服务功能，例如，剩余电量、存储卡容量提示和黑名单自动屏蔽等。通过这些自动服务功能，很好地为用户提供了人性化的服务，使整个操作过程更加方便。在本节的内容中，将通过几个典型实例的实现过程，来详细介绍这些自动服务功能的实现流程。

11.1　盟主的题目

太阳升起进入比赛的第二天。开始之前是盟主的例行讲话：“长江后浪推前浪，江山代有才人出！看到后起之秀们的精彩表现我很欣慰。现在言归正传，我们都知道江湖中消息埋伏很常见，当年的周伯通周老前辈在桃花阵中迷失了方向，锦毛鼠白玉堂血染冲霄楼……消息埋伏令很多名侠剑客头疼，其实无论什么消息埋伏，其目的都是用一种自动化装置来消灭对方。我们比武的第二关——消息埋伏的自动化，第二关题目都和自动化有关，凡是闯关成功者，一定会对消息埋伏有一个深刻的认识……”

11.2　短信自动提醒你

题　目	目　的	源码路径
题目 1	在手机中实现短信的自动提醒	“光盘:\daima\11\example1”文件夹

11.2.1　我的想法

我感受过短信自动提醒功能，如果收到了短信，系统会自动在手机上显示“有短信”的提示。Android 广播系统 BroadcastReseiver 能够实时监听手机的短信状况。当手机收到短信后，会通过 Notification 在状态栏中显示短信的摘要信息。我决定将接收到的短信对象解析为可以识别发信人号码和短信正文的字符串，这样结合 BroadcastReseiver 就能实现短信自动提醒功能。

11.2.2 具体实现

编写的主程序文件是 example1.java 和 example1_SMSreceiver.java，下面开始讲解其具体实现流程。

1. 文件 example1.java

文件 example1.java 功能是以 TextView 的文字显示“正在等待接收信息……”的提示。具体代码如下所示：

```
public class example1 extends Activity
{
  private TextView mTextView1;
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /*通过 findViewById 构造器创建 TextView 对象*/
    mTextView1 = (TextView) findViewById(R.id.myTextView1);
    mTextView1.setText("正在等待接收信息...");
  }
}
```

2. 文件 example1_SMSreceiver.java

(1) 自定义继承 BroadcastReceiver 类，用于聆听系统服务广播的信息。先声明静态字符串，并使用 android.provider.Telephony.SMS_RECEIVED 作为 Action 为短信的依据；然后通过 if 语句判断传来的 Intent 是否为短信，如果是，则先构建一字符串集合变量 sb，然后接收由 Intent 传来的数据；最后通过 if 语句判断 Intent 是否有数据。具体代码如下所示：

```
public class example1_SMSreceiver extends BroadcastReceiver
{
  /*声明静态字符串，并使用 android.provider.Telephony.SMS_RECEIVED
  作为 Action 为短信的依据*/
  private static final String mACTION =
  "android.provider.Telephony.SMS RECEIVED";

  @Override
  public void onReceive(Context context, Intent intent)
  {
    // TODO Auto-generated method stub
    /* 判断传来 Intent 是否为短信*/
    if (intent.getAction().equals(mACTION))
    {
      /*建构一字符串集合变量 sb*/
      StringBuilder sb = new StringBuilder();
      /*接收由 Intent 传来的数据*/
      Bundle bundle = intent.getExtras();
```

```
    /*判断 Intent 是否有数据*/
    if (bundle != null)
    {
      /* pdus 为 android 内置短信参数 identifier
       * 通过 bundle.get("")返回一包含 pdus 的对象*/
      Object[] myOBJpdus = (Object[]) bundle.get("pdus");
      /*构建短信对象 array，并依据收到的对象长度来创建 array 的大小*/
      SmsMessage[] messages = new SmsMessage[myOBJpdus.length];
      for (int i = 0; i<myOBJpdus.length; i++)
      {
        messages[i] =
        SmsMessage.createFromPdu((byte[]) myOBJpdus[i]);
      }

      /* 将送来的短信合并自定义信息于 StringBuilder 当中 */
      for (SmsMessage currentMessage : messages)
      {
        sb.append("正在接收到来自:\n");
        /* 来讯者的电话号码 */
        sb.append(currentMessage.getDisplayOriginatingAddress());
        sb.append("\n------发来的短信------\n");
        /* 取得传来信息的 BODY */
        sb.append(currentMessage.getDisplayMessageBody());
      }
    }
```

(2)　用 Notification(Toast)显示来讯信息，然后返回主 Activity 并设置让其以一个全新的 task 来运行。具体代码如下所示：

```
    Toast.makeText
    (
      context, sb.toString(), Toast.LENGTH_LONG
    ).show();

    /* 返回主 Activity */
    Intent i = new Intent(context, example1.class);
    /*设置让其以一个全新的 task 来运行*/
    i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    context.startActivity(i);
  }
 }
}
```

在文件 AndroidManifest.xml 中向系统注册常驻的 receiver，并设置这个 receiver 的 intent-filter 名为“android.provider.Telephony.SMS_RECEIVED”。另外，还需要设置 permission.RECEIVE_SMS 权限。具体代码如下所示：

```
  <!-- 建立 receiver 来聆听系统广播信息 -->
  <receiver android:name="example1_SMSreceiver">
  <!—设定要捕捉的讯息名称为 provider 中 Telephony.SMS_RECEIVED -->
 <intent-filter>
  <action
```

```
        android:name="android.provider.Telephony.SMS RECEIVED" />
    </intent-filter>
      </receiver>
</application>
<uses-permission
android:name="android.permission.RECEIVE_SMS"></uses-permission>
```

至此整个任务完成，执行后的初始效果如图 11-1 所示。当接收到短信后会在屏幕中显示对应的提示信息，如图 11-2 所示。



图 11-1　初始效果



图 11-2　短信提示

同时在系统的短信栏目中会显示收到的短信，如图 11-3 所示。



图 11-3　收到的短信

注意： 在具体测试时，我们需要同时运行两个模拟器，一个用于发送短信；另一个用于接收短信，但是如果机器太慢，无法启动两个模拟器，也可以只启动一个模拟器。然后在 Eclipse 菜单中依次单击 windows | show view | other | Android | Emulator Control 菜单命令，打开 Emulator Control 面板。在 Telephony Actions 分组栏中，Voice 是呼叫，SMS 是发送短信。Incoming number 是模拟器的端口号，我们也可以使用这个功能给模拟器拨打电话或发送短信，Emulator Control 面板如图 11-4 所示。

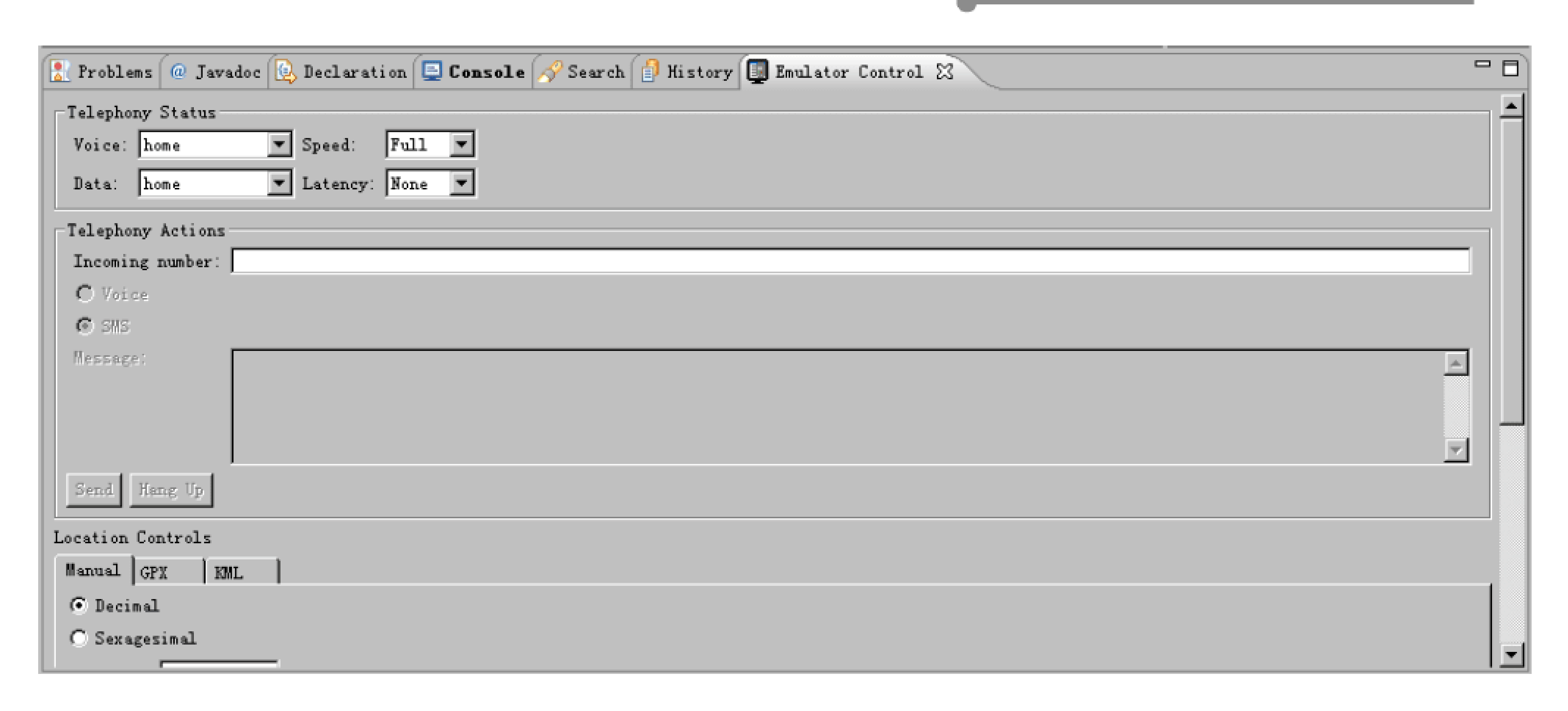


图 11-4　Emulator Control 面板

11.3　电池容量剩几何

题　目	目　的	源码路径
题目 2	实现查看手机电池的容量	“光盘:\daima\11\example2”文件夹

11.3.1　我的想法

在使用手机的过程中，最担心的是害怕手机没电而影响业务，所以在屏幕上及时显示电池容量就很有必要了。《Android SDK 4.0 密录》中写道：可以使用 BroadcastReceiver 的特性来获取手机电池的容量，通过注册 BroadcastReseiver 时设置的 IntentFilter 来获取系统发出的 Intent.ACTION_BATTERY_CHANGED，然后以此来获取电池的容量。

11.3.2　具体实现

编写的主程序文件是 example2.java，下面开始讲解其具体实现代码。

(1) 分别声明三个变量 intLevel、intScale 和 mButton01，然后创建 BroadcastReceiver，如果捕捉到的 Action 是 ACTION_BATTERY_CHANGED，则运行 onBatteryInfoReceiver()。具体代码如下所示：

```
public class example2 extends Activity
{
  /* 变量声明 */
  private int intLevel;
  private int intScale;
  private Button mButton01;

  /* 创建 BroadcastReceiver */
  private BroadcastReceiver mBatInfoReceiver=new BroadcastReceiver()
  {
```

```
  public void onReceive(Context context, Intent intent)
  {
    String action = intent.getAction();
    /* 如果捕捉到的 action 是 ACTION_BATTERY_CHANGED,
     * 就运行 onBatteryInfoReceiver() */
    if (Intent.ACTION_BATTERY_CHANGED.equals(action))
    {
      intLevel = intent.getIntExtra("level", 0);
      intScale = intent.getIntExtra("scale", 100);
      onBatteryInfoReceiver(intLevel,intScale);
    }
  }
};
```

(2) 在 onCreate 中载入主布局文件 main.xml，然后初始化 Button 和设置单击后的动作，并注册一个系统 BroadcastReceiver，用于访问电池计量。具体代码如下所示：

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  /* 载入 main.xml Layout */
  setContentView(R.layout.main);

  /* 初始化 Button，并设置单击后的动作 */
  mButton01 = (Button)findViewById(R.id.myButton1);
  mButton01.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View v)
    {
      /* 注册一个系统 BroadcastReceiver，作为访问电池计量之用 */
      registerReceiver
      (
        mBatInfoReceiver,
        new IntentFilter(Intent.ACTION_BATTERY_CHANGED)
      );
    }
  });
}
```

(3) 定义方法 onBatteryInfoReceiver，当捕捉到 ACTION_BATTERY_CHANGED 时要运行的 Method 时，首先创建一个背景模糊的 Window 且将对话框放在前景，然后将取得的电池计量显示于 Dialog 中，最后设置返回主画面的按钮。具体代码如下所示：

```
/* 捕捉到 ACTION_BATTERY_CHANGED 时要运行的 method */
public void onBatteryInfoReceiver(int intLevel, int intScale)
{
  /* create 跳出的对话框 */
  final Dialog d = new Dialog(example2.this);
  d.setTitle(R.string.str_dialog_title);
```

```
    d.setContentView(R.layout.mydialog);

    /* 创建一个背景模糊的 Window，且将对话框放在前景 */
    Window window = d.getWindow();
    window.setFlags
    (
      WindowManager.LayoutParams.FLAG_BLUR_BEHIND,
      WindowManager.LayoutParams.FLAG_BLUR_BEHIND
    );

    /* 将取得的电池计量显示于 Dialog 中 */
    TextView mTextView02=(TextView)d.findViewById(R.id.myTextView2);
    mTextView02.setText
    (
      getResources().getText(R.string.str_dialog_body)+
      String.valueOf(intLevel * 100 / intScale) + "%"
    );

    /* 设置返回主画面的按钮 */
    Button mButton02 = (Button)d.findViewById(R.id.myButton2);
    mButton02.setOnClickListener(new Button.OnClickListener()
    {
      @Override
      public void onClick(View v)
      {
        /* 反注册 Receiver，并关闭对话框 */
        unregisterReceiver(mBatInfoReceiver);
        d.dismiss();
      }
    });
    d.show();
  }
}
```

至此整个任务完成，执行后的初始效果如图 11-5 所示。当单击“获取”按钮后会显示当前电池的容量，容量显示如图 11-6 所示。

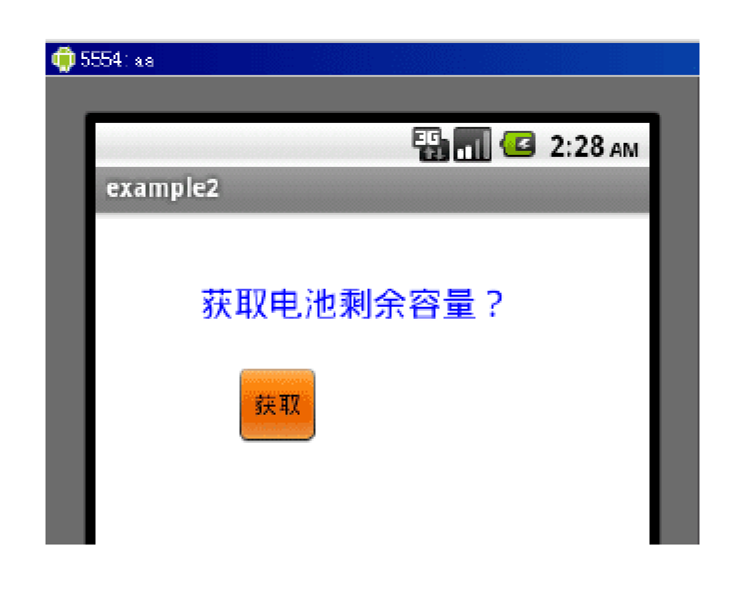


图 11-5　初始效果



图 11-6　容量显示

11.4 群发英雄帖

题 目	目 的	源码路径
题目 3	编程实现短信群发功能	“光盘:\daima\11\example3”文件夹

11.4.1 我的想法

我准备预先插入一个按钮，当单击“发送”按钮后，会首先获取手机通讯录的信息，让用户选择短信接收者，选好后返回主程序，然后实现短信群发功能。当然需要在通讯录中添加一些联系人信息，这些联系人作为短信接收者。当用户选择接收者后，会将短信发送到目标者。

11.4.2 具体实现

编写的主文件为 example3.java，其具体流程如下。

(1) 分别设置两个 Button 的处理事件，其中 mButton01 用于获取 mTextView3 中的内容，mButton02 用于获取 mTextView5 中的内容。具体代码如下所示：

```
/*设置第一个 Button 事件*/
mButton01.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    // TODO Auto-generated method stub

    Uri uri = Uri.parse("content://contacts/people");
    Intent intent = new Intent(Intent.ACTION_PICK, uri);
    /*获取 mTextView3 里的内容*/
    strMessage = mTextView3.getText().toString();

    startActivityForResult(intent, PICK_CONTACT_SUBACTIVITY);
  }
});

/*设置第二个 Button 事件*/
mButton02.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    // TODO Auto-generated method stub
    Uri uri = Uri.parse("content://contacts/people");
    Intent intent = new Intent(Intent.ACTION_PICK, uri);
    /*获取 mTextView5 里的内容*/
    strMessage = mTextView11.getText().toString();
```

```
        startActivityForResult(intent, PICK CONTACT SUBACTIVITY);
      }
    });
  }
```

(2) 在获取 android.permission.READ_CONTACTS 的权限下，通过 try 来抓取通讯录的姓名、电话，然后在通讯录中选取接收者的电话号码，接着用 smsManager.sendTextMessage 发送短信，最后用 Toast 来显示正在传送的提示。具体代码如下所示：

```
@Override
protected void onActivityResult
(int requestCode, int resultCode, Intent data)
{
  // TODO Auto-generated method stub
  switch (requestCode)
  {
    case PICK_CONTACT_SUBACTIVITY:
      final Uri uriRet = data.getData();
      if(uriRet != null)
      {
        try
        {
          /* 必须要有 android.permission.READ_CONTACTS 权限 */
          Cursor c = managedQuery(uriRet, null, null, null, null);
          c.moveToFirst();
          /*抓取通讯录的姓名*/
          String strName =
          c.getString(c.getColumnIndexOrThrow(People.NAME));
          /*抓取通讯录的电话*/
          String strPhone =
          c.getString(c.getColumnIndexOrThrow(People.NUMBER));

          /*设置要寄给通讯录里的电话*/
          String strDestAddress = strPhone;
          System.out.println(strMessage);
          SmsManager smsManager = SmsManager.getDefault();

          PendingIntent mPI = PendingIntent.getBroadcast
          (example3.this, 0, new Intent(), 0);
          /*寄出短信*/
          smsManager.sendTextMessage
          (
            strDestAddress, null, strMessage, mPI, null
          );
          /*用 Toast 显示传送中*/
          Toast.makeText
          (
            example3.this,
            getString(R.string.str_msg)+strName,
            Toast.LENGTH_SHORT
          ).show();
```

```
          mTextView01.setText(strName+":"+strPhone);
         }
         catch(Exception e)
         {
           mTextView01.setText(e.toString());
           e.printStackTrace();
         }
       }
       break;
   }
   super.onActivityResult(requestCode, resultCode, data);
 }
}
```

在 AndroidManifest.xml 中设置允许 READ_CONTACTS 权限和 SEND_SMS 权限。主要代码如下：

```
<uses-permission
android:name="android.permission.READ_CONTACTS"></uses-permission>
<uses-permission
android:name="android.permission.SEND_SMS"></uses-permission>
```

至此整个任务完成，执行后的初始效果如图 11-7 所示，当单击“发送”按钮后会显示联系人界面，如图 11-8 所示。

单击其中的一个联系人后会将短信发送给他，并输出发送提示，如图 11-9 所示。



图 11-7 初始效果

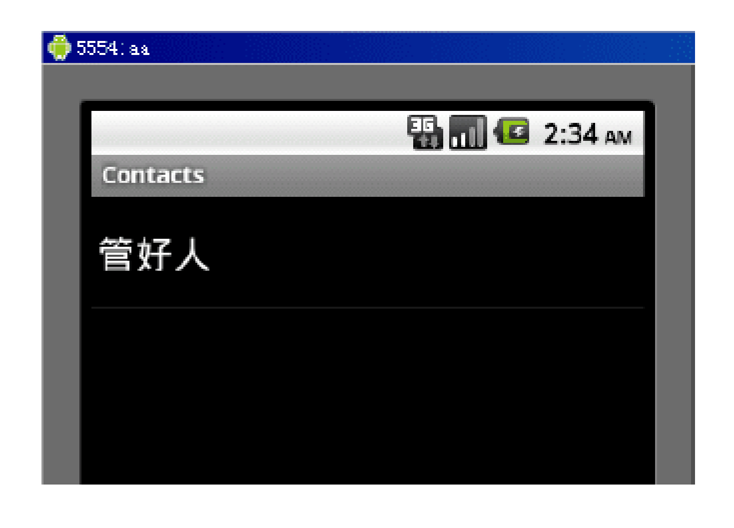


图 11-8 联系人界面



图 11-9 已发送

注意： 在上述实例中，虽然实现了短信的发送功能，但是还不能算是群组发送。实际上 Android 系统允许用户创建若干个群组。可以把你的联系人很轻松地丢进各种不同的群组里。Android 系统之所以提供了这样的一个特性，是为了让用户可以给整组联系人群发邮件或者短信，这是一个很贴心的功能，如图 11-10 所示。

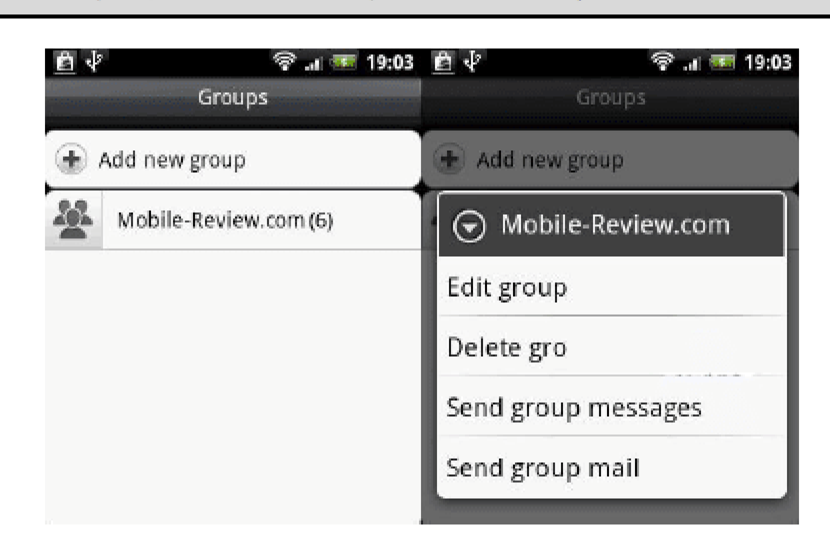


图 11-10 Android 的群组

另外，也可以用编程的方式实现群发短信。既可以把联系人的数据用字符串数组的形式保存，也可以以 cursor 为对象来通过循环的方式，当取得了联系人数据的同时就传出指定的内容。

11.5　来电提醒

题　目	目　的	源码路径
题目 4	如果有来电则会在屏幕中显示拨打用户的基本信息	“光盘:\daima\11\example5”文件夹

11.5.1　我的想法

我知道在当前手机应用中，如果有来电则会在屏幕中显示拨打用户的姓名等基本信息。但是从没有做过类似的功能，无奈之下只好翻开《Android SDK 4.0 密录》，上面写道：在 Android 中，可以通过 PhoneStateListener 提供的方法来监听来电状态。在具体实施时，需要创建 PhoneStateListener 对象，并重写其中的 onCallStateChanged()方法，并通过传入的 state 来判断来电状态。要获取来电状态，需要用户读取电话状态的权限，否则不能成功获取状态。在具体实施上，需要在模拟器中先添加一个联系人记录并为其起名。这样当电话打进来后，会在屏幕中显示它的名字。如果是非通讯录的来电，则在屏幕中显示 Unknown Caller。

11.5.2　具体实现

编写的主程序文件是 example5.java，其具体实现流程如下。

(1)　通过 setContentView 来引用主布局文件 main.xml，然后通过 myTextView1 显示提示。具体代码如下所示：

```
public class example5 extends Activity
{
  private TextView myTextView1;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    myTextView1 = (TextView) findViewById(R.id.myTextView1);
```

(2)　定义 TelephonyManager 对象 tm，用于获取电话服务，然后通过 tm.listen 来注册电话通信 Listener。具体代码如下所示：

```
    /* 添加自己实现的 PhoneStateListener */
    exPhoneCallListener myPhoneCallListener =
    new exPhoneCallListener();

    /* 取得电话服务 */
```

```
  TelephonyManager tm = (TelephonyManager) this.getSystemService (Context.
  TELEPHONY_SERVICE);

  /* 注册电话通信 Listener */
  tm.listen
  (
    myPhoneCallListener,
    PhoneStateListener.LISTEN_CALL_STATE
  );
}
```

(3) 使用内部 class 来继承 PhoneStateListener，重写 onCallStateChanged，这样当状态改变时改变 myTextView1 的文字及颜色，然后分别设置无任何状态、接起电话、电话进来时的显示。具体代码如下所示：

```
public class exPhoneCallListener extends PhoneStateListener
{
  /* 重写 onCallStateChanged
  当状态改变时改变 myTextView1 的文字及颜色 */
  public void onCallStateChanged(int state, String incomingNumber)
  {
    switch (state)
    {
      /* 无任何状态时 */
      case TelephonyManager.CALL_STATE_IDLE:
        myTextView1.setTextColor
        (
          getResources().getColor(R.drawable.red)
        );
        myTextView1.setText("CALL_STATE_IDLE");
        break;
      /* 接起电话时 */
      case TelephonyManager.CALL_STATE_OFFHOOK:
        myTextView1.setTextColor
        (
          getResources().getColor(R.drawable.green)
        );
        myTextView1.setText("CALL_STATE_OFFHOOK");
        break;
      /* 电话进来时 */
      case TelephonyManager.CALL_STATE_RINGING:
        getContactPeople(incomingNumber);
        break;
      default:
        break;
    }
    super.onCallStateChanged(state, incomingNumber);
  }
}
```

(4) 通过方法 getContactPeople(String incomingNumber)来获取机器内的联系人信息，然后在 cursor 里存放要放的字段名称。具体代码如下所示：

```
private void getContactPeople(String incomingNumber)
{
  myTextView1.setTextColor(Color.BLUE);
  ContentResolver contentResolver = getContentResolver();
  Cursor cursor = null;
  /* cursor 里要放的字段名称 */
  String[] projection = new String[]
  {
    Contacts.People._ID,
    Contacts.People.NAME,
    Contacts.People.NUMBER
  };
```

(5) 通过来电号码来查找对应的联系人，查找到则显示姓名，没有查找到则只显示号码。具体代码如下所示：

```
  /* 用来电电话号码去查找该联系人 */
  cursor = contentResolver.query
  (
    Contacts.People.CONTENT_URI, projection,
    Contacts.People.NUMBER + "=?",
    new String[]
    {
      incomingNumber
    },
    Contacts.People.DEFAULT_SORT_ORDER
  );
  /* 找不到联系人 */
  if (cursor.getCount() == 0)
  {
    myTextView1.setText("unknown Number:" + incomingNumber);
  }
  else if (cursor.getCount() > 0)
  {
    cursor.moveToFirst();
    /* 在 projection 这个数组里名字放在第 1 个位置 */
    String name = cursor.getString(1);
    myTextView1.setText(name + ":" + incomingNumber);
  }
 }
}
```

还需要在文件 AndroidManifest.xml 中获取如下两个权限：

```
<uses-permission
android:name="android.permission.READ_CONTACTS"></uses-permission>
<uses-permission
android:name="android.permission.READ_PHONE_STATE"></uses-permission>
```

读取通讯录权限：android.permission.READ_CONTACTS。
获取电话状态权限：android.permission.READ_PHONE_STATE。
至此整个任务完成，执行后的初始效果如图 11-11 所示，当打来电话后会显示来电的基本

信息，如图 11-12 所示。



图 11-11　初始效果



图 11-12　来电信息

11.6　存储卡容量有几何

题　目	目　的	源码路径
题目 5	编程实现获取存储卡容量	“光盘:\daima\11\example6”文件夹

11.6.1　我的想法

在手机应用中同样需要及时了解存储卡的容量信息。我知道存储卡是可以随时插拔的，每次插拔时会对操作系统进行 ACTION broadcast 设置。经过仔细权衡之下，我决定使用 StatFs 文件系统来获取 MicroSD 存储卡的剩余容量。在具体实施时，首先判断是否安装存储卡，如果不存在则直接不计算。为了更好地显示容量，在布局中插入了一个 ProgressBar Widget，这样将通过图形界面将容量显示得更加一目了然。

11.6.2　具体实现

编写的主程序文件是 example6.java，其具体实现流程如下。

(1)　定义 setOnClickListener，用于触发按钮单击事件处理程序。具体代码如下所示：

```
  myButton.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View arg0)
    {
      showSize();
    }
  });
}
```

(2)　定义方法 showSize()用于显示存储卡的容量大小。具体代码如下所示：

```
private void showSize()
{
  /*  将 TextView 及 ProgressBar 设置为空值及 0 */
  myTextView.setText("");
  myProgressBar.setProgress(0);
  /* 判断存储卡是否插入 */
  if
(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED))
  {
    /*   取得 SD CARD 文件路径一般是/sdcard */
    File path = Environment.getExternalStorageDirectory();
    /* StatFs 看文件系统空间使用状况 */
    StatFs statFs = new StatFs(path.getPath());
    /* Block 的 size*/
    long blockSize = statFs.getBlockSize();
    /* 总 Block 数量 */
    long totalBlocks = statFs.getBlockCount();
    /*   已使用的 Block 数量 */
    long availableBlocks = statFs.getAvailableBlocks();
    String[] total = fileSize(totalBlocks * blockSize);
    String[] available = fileSize(availableBlocks * blockSize);
    /* getMax 取得在 main.xml 里 ProgressBar 设置的最大值 */
    int ss = Integer.parseInt(available[0]) * myProgressBar.getMax()
        / Integer.parseInt(total[0]);
    myProgressBar.setProgress(ss);
    String text = "总共" + total[0] + total[1] + "\n";
    text += "可用" + available[0] + available[1];
    myTextView.setText(text);
  } else if (Environment.getExternalStorageState().equals(
      Environment.MEDIA_REMOVED))
  {
    String text = "SD CARD 已删除";
    myTextView.setText(text);
  }
}
/*返回为字符串数组[0]为大小[1]为单位 KB 或 MB*/
private String[] fileSize(long size)
{
  String str = "";
  if (size >= 1024)
  {
    str = "KB";
    size /= 1024;
    if (size >= 1024)
    {
      str = "MB";
      size /= 1024;
    }
  }
  DecimalFormat formatter = new DecimalFormat();
  /* 每 3 个数字用“,”分隔如：1,000 */
  formatter.setGroupingSize(3);
```

```
    String result[] = new String[2];
    result[0] = formatter.format(size);
    result[1] = str;
    return result;
  }
}
```

至此整个任务完成，执行后的效果如图 11-13 所示。



图 11-13　执行后的效果

Android 模拟器能够让我们使用 FAT32 格式的磁盘镜像作为 SD 卡的模拟，具体过程如下。

(1)　进入 Android SDK 目录下的 tools 子目录，运行如下的代码：

```
mksdcard -l sdcard 512M /your_path_for_img/sdcard.img
```

这样就创建了一个 512MB 的 SD 卡镜像文件。

(2)　运行模拟器的时候指定路径(注意需要完整路径)：

```
emulator -sdcard /your_path_for_img/sdcard.img
```

这样模拟器中就可以使用“/sdcard”这个路径来指向模拟的 SD 卡了。

那么如何复制本机文件到 SD 卡，或者管理 SD 卡上的内容呢？有如下两种方案。

第一种：在 Linux 下面我们可以 mount 成一个 loop 设备，先创建一个目录比如叫 android_sdcard，然后执行下面的代码：

```
mount -o loop sdcard.img android_sdcard
```

这样管理这个目录就是管理 sdcard 的内容了。

第二种：在 windows 可视环境下我们也可以用 mtools 来做管理，也可以用 android SDK 自带的命令(这个命令在 Linux 下面也可以用)：

```
adb push local_file sdcard/remote_file
```

在使用 mksdcard 命令时要注意如下 6 点。

(1)　mycard 命令可以使用三种单位：Byte(字节)、KB 和 MB。如果只使用数字表示字节，后面还可以跟 KB，如 262144KB，也表示 256MB。

(2)　mycard 建立的虚拟文件最小为 8MB，也就是说，模拟器只支持大于 8MB 的虚拟文件。

(3)　-l 命令行参数表示虚拟磁盘的卷标，也可以没有该参数。

(4)　虚拟文件的扩展名可以是任意的，如 mycard.abc。

(5)　mksdcard 命令不会自动建立不存在的目录，因此，在执行上面的命令之前，要先在当前目录中建立一个 card 目录。

(6)　mksdcard 命令按实际大小生成 sdcard 虚拟文件。例如，生成 256MB 虚拟文件的大小就

是 256MB，如果生成较大的虚拟文件，要确保硬盘是否有足够空间。

在执行完上面的命令后，可以执行下面的命令启动 Android 模拟器：

```
emulator -avd avd1 -sdcard card/mycard.img
```

如果在 Eclipse 开发环境中，可以在 Run Configuration 对话框中设置启动参数，当然也可以在 Preferences 对话框中设置默认启动参数。这样在新建立的 Android 工程中就自动加入了装载 sdcard 虚拟文件的命令行参数。

如果读者使用 OPhone 虚拟机，设置的方法也是完全一样的，然后在虚拟机中的 Setting 里看看 sdcard 是否找到。那么如何查看 sdcard 虚拟设备中的内容呢？方法很多，最简单的就是使用 Android Eclipse 插件自带的 DDMS 透视图。

11.7　内存和存储卡控制

题　目	目　的	源码路径
题目 6	对内存和存储卡中文件进行操作	“光盘:\daima\11\example8”文件夹

11.7.1　我的想法

移动手机的存储控件分为内存控件和存储卡控件，我决定在屏幕中添加两个按钮，分别用于添加和删除内存或存储卡内的文件，并且在实例中使用了 3 个 Activity，主程序的是 Entry Activity，另外两个分别用于处理内存卡和存储卡。当用户选择内存或存储卡后，将以列表形式显示里面的所有目录和文件名，并在 MENU 菜单中显示“添加”或“删除”按钮。单击“添加”按钮后会显示一个添加菜单，实现添加文件的功能。当单击“删除”按钮后，可以删除指定的文件。

11.7.2　具体实现

编写的主程序文件是 example8.java、example8_1.java 和 example8_2.java。

1. 文件 example8.java

(1) 通过 getFilesDir()方法取得 SD Card 目录，设置当 SD Card 无插入时 myButton2 处于不能按状态。具体代码如下所示：

```
    /* 取得目前 File 目录 */
    fileDir = this.getFilesDir();
    /* 取得 SD Card 目录 */
    sdcardDir = Environment.getExternalStorageDirectory();
    /* 当 SD Card 无插入时将 myButton2 设成不能按 */
    if
(Environment.getExternalStorageState().equals(Environment.MEDIA_REMOVED))
    {
      myButton2.setEnabled(false);
    }
```

(2) 分别定义按钮单击处理事件 myButton1.setOnClickListener 和 myButton2.setOnClickListener。具体代码如下所示：

```
  myButton1.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View arg0)
    {
      String path = fileDir.getParent() + java.io.File.separator
          + fileDir.getName();
      showListActivity(path);
    }
  });
  myButton2.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View arg0)
    {
      String path = sdcardDir.getParent() + sdcardDir.getName();
      showListActivity(path);
    }
  });
}
```

(3) 采用方法 showListActivity(String path)定义一个 Intent 对象 intent，然后将路径传到 example_1。具体代码如下所示：

```
  private void showListActivity(String path)
  {
    Intent intent = new Intent();
    intent.setClass(example8.this, example8_1.class);
    Bundle bundle = new Bundle();
    /* 将路径传到 example_1 */
    bundle.putString("path", path);
    intent.putExtras(bundle);
    startActivity(intent);
  }
}
```

2. 文件 example8_1.java

(1) 将主 Activity 传来的 path 字符串作为传入路径，如果路径不在，则使用 java.io.File 来创建。具体代码如下所示：

```
public class example8_1 extends ListActivity
{
  private List<String> items = null;
  private String path;
  protected final static int MENU_NEW = Menu.FIRST;
  protected final static int MENU DELETE = Menu.FIRST + 1;
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.ex06_09_1);
    Bundle bunde = this.getIntent().getExtras();
    path = bunde.getString("path");
    java.io.File file = new java.io.File(path);
    /* 当该目录不存在时将目录创建 */
    if (!file.exists())
    {
      file.mkdir();
    }
    fill(file.listFiles());
  }
```

(2) 定义 onOptionsItemSelected，根据单击的 MENU 按钮实现添加或删除。具体代码如下所示：

```
  @Override
  public boolean onOptionsItemSelected(MenuItem item)
  {
    super.onOptionsItemSelected(item);
    switch (item.getItemId())
    {
      case MENU_NEW:
        /* 单击添加 MENU */
        showListActivity(path, "", "");
        break;
      case MENU_DELETE:
        /* 单击删除 MENU */
        deleteFile();
        break;
    }
    return true;
  }
```

(3) 定义方法 onCreateOptionsMenu(Menu menu)，用于添加需要的 MENU。具体代码如下所示：

```
  @Override
  public boolean onCreateOptionsMenu(Menu menu)
  {
    super.onCreateOptionsMenu(menu);
    /* 添加 MENU */
    menu.add(Menu.NONE, MENU_NEW, 0, R.string.strNewMenu);
    menu.add(Menu.NONE, MENU_DELETE, 0, R.string.strDeleteMenu);
    return true;
  }
```

(4) 当单击文件名后取得文件内容。具体代码如下所示：

```
  @Override
  protected void onListItemClick
  (ListView l, View v, int position, long id)
  {
```

```
  File file = new File
  (path + java.io.File.separator + items.get(position));

  /* 单击文件取得文件内容 */
  if (file.isFile())
  {
    String data = "";
    try
    {
      FileInputStream stream = new FileInputStream(file);
      StringBuffer sb = new StringBuffer();
      int c;
      while ((c = stream.read()) != -1)
      {
        sb.append((char) c);
      }
      stream.close();
      data = sb.toString();
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    showListActivity(path, file.getName(), data);
  }
}
```

(5) 定义 fill(File[] files)，用于填充内容到文件。具体代码如下所示：

```
private void fill(File[] files)
{
  items = new ArrayList<String>();
  if (files == null)
  {
    return;
  }
  for (File file : files)
  {
    items.add(file.getName());
  }
  ArrayAdapter<String> fileList = new ArrayAdapter<String>
  (this,android.R.layout.simple_list_item_1, items);
  setListAdapter(fileList);
}
```

(6) 定义 showListActivity，用于显示已经存在的文件列表。具体代码如下所示：

```
private void showListActivity
(String path, String  ilename, String data)
{
  Intent intent = new Intent();
  intent.setClass(example8 1.this, example8 2.class);
```

```
    Bundle bundle = new Bundle();
    /* 文件路径 */
    bundle.putString("path", path);
    /* 文件名 */
    bundle.putString("ilename",  ilename);
    /* 文件内容 */
    bundle.putString("data", data);
    intent.putExtras(bundle);
    startActivity(intent);
  }
```

(7) 定义 deleteFile()，用于删除选定的文件。具体代码如下所示：

```
  private void deleteFile()
  {
    int position = this.getSelectedItemPosition();
    if (position >= 0)
    {
      File file = new File(path + java.io.File.separator +
      items.get(position));

      /* 删除文件 */
      file.delete();
      items.remove(position);
      getListView().invalidateViews();
    }
  }
}
```

3. 文件 example8_2.java

当单击“添加”按钮后会来到 example8_2.java，其具体实现流程如下。

(1) 设置 myEditText1，用于放置文件内容，然后定义 Bundle 对象 bunde，用于获取路径 path 和数据 data。具体代码如下所示：

```
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.ex06_09_2);
    /* 放置文件内容的 EditText */
    myEditText1 = (EditText) findViewById(R.id.myEditText1);

    Bundle bunde = this.getIntent().getExtras();
    path = bunde.getString("path");
    data = bunde.getString("data");
    fileName = bunde.getString("fileName");
    myEditText1.setText(data);
  }
```

(2) 使用 onOptionsItemSelected 根据用户选择而进行操作。当选择 MENU_SAVE 时，保存这个文件。具体代码如下所示：

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
  super.onOptionsItemSelected(item);
  switch (item.getItemId())
  {
    case MENU_SAVE:
      saveFile();
      break;
  }
  return true;
}
```

(3) 定义 onCreateOptionsMenu(Menu menu)，用于添加一个 MENU 按钮。具体代码如下所示：

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
  super.onCreateOptionsMenu(menu);
  /* 添加 MENU */
  menu.add(Menu.NONE, MENU_SAVE, 0, R.string.strSaveMenu);
  return true;
}
```

(4) 定义 saveFile()，用于保存文件。先定义 LayoutInflater 对象 factory，用于跳出存档，然后通过 myDialogEditText 取得 Dialog 里的 EditText，最后通过实现存档处理。具体代码如下所示：

```
private void saveFile()
{
  /* 跳出存档的 Dialog */
  LayoutInflater factory = LayoutInflater.from(this);

  final View textEntryView = factory.inflate
  (R.layout.save_dialog, null);

  Builder mBuilder1 = new AlertDialog.Builder(example8_2.this);

  mBuilder1.setView(textEntryView);
  /* 取得 Dialog 里的 EditText */
  myDialogEditText = (EditText) textEntryView.findViewById
                                (R.id.myDialogEditText);

  myDialogEditText.setText(fileName);

  mBuilder1.setPositiveButton
  (
```

```
    R.string.str_alert_ok,new DialogInterface.OnClickListener()
    {
      public void onClick(DialogInterface dialoginterface, int i)
      {
        /* 存档 */
        String Filename = path + java.io.File.separator
            + myDialogEditText.getText().toString();
        java.io.BufferedWriter bw;
        try
        {
          bw = new java.io.BufferedWriter(new java.io.FileWriter(
              new java.io.File(Filename)));
          String str = myEditText1.getText().toString();
          bw.write(str, 0, str.length());
          bw.newLine();
          bw.close();
        }
        catch (IOException e)
        {
          e.printStackTrace();
        }
        /* 回到example8_1 */
        Intent intent = new Intent();
        intent.setClass(example8_2.this, example8_1.class);
        Bundle bundle = new Bundle();
        /* 将路径传到example8_1 */
        bundle.putString("path", path);
        intent.putExtras(bundle);
        startActivity(intent);
        finish();
      }
    });
  mBuilder1.setNegativeButton(R.string.str_alert_cancel, null);
  mBuilder1.show();
 }
}
```

至此整个任务完成。执行后的初始效果如图 11-14 所示，当单击一个按钮后会显示对应的存储信息，如图 11-15 所示。此时单击 MENU 按钮后会弹出两个 Menu 选项，如图 11-16 所示。此时，可以通过这两个选项分别对存储卡中的数据实现管理。



图 11-14　初始效果



图 11-15　SD 卡的文件信息



图 11-16　管理 Menu

11.8　闹钟的提醒

题　目	目　的	源码路径
题目 7	在 Android 中实现定时闹钟	“光盘:\daima\11\example9”文件夹

11.8.1　我的想法

闹钟对我并不陌生，在《Android SDK 4.0 密录》中写道：可以用 AlarmManager 类设置在指定时间运行某些动作，并且在 Android 中内置了 Alarm Clock，可以实现闹钟的功能。

11.8.2　具体实现

编写的主程序文件是 example9.java、example9_1.java 和 example9_2.java。

1. 文件 example9.java

(1)　载入主布局文件 main.xml，首先通过 setTime1 实现只响一次的闹钟设置，然后实现只响一次的闹钟设置 Button1。具体代码如下所示：

```
@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  /* 载入 main.xml Layout */
  setContentView(R.layout.main);
```

```
/* 以下为只响一次的闹钟的设置 */
setTime1=(TextView) findViewById(R.id.setTime1);
/* 只响一次的闹钟的设置 Button */
mButton1=(Button)findViewById(R.id.mButton1);
mButton1.setOnClickListener(new View.OnClickListener()
{
  public void onClick(View v)
  {
    /* 取得点击按钮时的时间作为 TimePickerDialog 的默认值 */
    c.setTimeInMillis(System.currentTimeMillis());
    int mHour=c.get(Calendar.HOUR OF DAY);
    int mMinute=c.get(Calendar.MINUTE);
```

(2) 通过 TimePickerDialog 来弹出一个对话框供用户来设置时间，具体实现流程如下。

第 1 步：获取设置后的时间，并指定闹钟设置时间到时要运行 CallAlarm.class。

第 2 步：创建 PendingIntent 对象 sender，开始加载 example9 的 intent。

第 3 步：通过 AlarmManager.RTC_WAKEUP 设置服务在系统休眠时同样会运行。

第 4 步：定义 tmpS，用于更新显示设置的闹钟时间并以 Toast 提示设置已完成。

具体代码如下所示：

```
    /* 跳出 TimePickerDialog 来设置时间 */
    new TimePickerDialog(example9.this,
      new TimePickerDialog.OnTimeSetListener()
      {
        public void onTimeSet(TimePicker view,int hourOfDay,
                              int minute)
        {
          /* 取得设置后的时间，秒跟毫秒设为 0 */
          c.setTimeInMillis(System.currentTimeMillis());
          c.set(Calendar.HOUR_OF_DAY,hourOfDay);
          c.set(Calendar.MINUTE,minute);
          c.set(Calendar.SECOND,0);
          c.set(Calendar.MILLISECOND,0);

          /* 指定闹钟设置时间到时要运行 CallAlarm.class */
          Intent intent = new Intent(example9.this, example9_2.class);
          /* 创建 PendingIntent */
          PendingIntent sender=PendingIntent.getBroadcast(
                                   example9.this,0, intent, 0);
          /* AlarmManager.RTC_WAKEUP 设置服务在系统休眠时同样会运行
           * 以 set()设置的 PendingIntent 只会运行一次
           * */
          AlarmManager am;
          am = (AlarmManager)getSystemService(ALARM_SERVICE);
          am.set(AlarmManager.RTC_WAKEUP,
                c.getTimeInMillis(),
                sender
               );
          /* 更新显示的设置闹钟时间 */
```

```
            String tmpS=format(hourOfDay)+": "+format(minute);
            setTime1.setText(tmpS);
            /* 以 Toast 提示设置已完成 */
            Toast.makeText(example9.this,"设置闹钟时间为"+tmpS,
              Toast.LENGTH_SHORT)
              .show();
          }
        },mHour,mMinute,true).show();
    }
  });
```

(3) 设置按钮 mButton2，用于实现只响一次的闹钟的删除。首先在 AlarmManager 中实现删除，然后通过 Toast 提示已删除设置并更新显示的闹钟时间。具体代码如下所示：

```
/* 只响一次的闹钟的删除 Button */
mButton2=(Button) findViewById(R.id.mButton2);
mButton2.setOnClickListener(new View.OnClickListener()
{
  public void onClick(View v)
  {
    Intent intent = new Intent(example9.this, example9_2.class);
    PendingIntent sender=PendingIntent.getBroadcast(
                        example9.this,0, intent, 0);
    /* 由 AlarmManager 中删除 */
    AlarmManager am;
    am =(AlarmManager)getSystemService(ALARM_SERVICE);
    am.cancel(sender);
    /* 以 Toast 提示已删除设置，并更新显示的闹钟时间 */
    Toast.makeText(example9.this,"闹钟时间解除",
                Toast.LENGTH_SHORT).show();
    setTime1.setText("目前无设置");
  }
});
```

(4) 设置重复响起的闹钟，具体实现流程如下。

第 1 步：以 create 重复响起闹钟的设置画面，并引用 timeset.xml 为布局文件。

第 2 步：以 create 重复响起闹钟的设置 Dialog 对话框。

第 3 步：获取设置的间隔秒数和设置的开始时间，秒及毫秒都设为 0。

第 4 步：指定闹钟设置时间到时要运行 CallAlarm.class。

第 5 步：通过 setRepeating()可以让闹钟重复运行，通过 tmpS 更新显示的设置闹钟时间。

第 6 步：通过 Toast 提示设置已完成。

具体代码如下所示：

```
/* 以下为重复响起的闹钟的设置 */
setTime2=(TextView) findViewById(R.id.setTime2);
/* create 重复响起的闹钟的设置画面 */
/* 引用 timeset.xml 为 Layout */
LayoutInflater factory = LayoutInflater.from(this);
final View setView = factory.inflate(R.layout.timeset,null);
final TimePicker tPicker=(TimePicker)setView
                        .findViewById(R.id.tPicker);
```

```
tPicker.setIs24HourView(true);

/* create 重复响起闹钟的设置 Dialog */
final AlertDialog di=new AlertDialog.Builder(example9.this)
      .setIcon(R.drawable.clock)
      .setTitle("设置")
      .setView(setView)
      .setPositiveButton("确定",
        new DialogInterface.OnClickListener()
       {
         public void onClick(DialogInterface dialog, int which)
         {
           /* 取得设置的间隔秒数 */
           EditText ed=(EditText)setView.findViewById(R.id.mEdit);
           int times=Integer.parseInt(ed.getText().toString())
                    *1000;
           /* 取得设置的开始时间，秒及毫秒设为 0 */
           c.setTimeInMillis(System.currentTimeMillis());
           c.set(Calendar.HOUR_OF_DAY,tPicker.getCurrentHour());
           c.set(Calendar.MINUTE,tPicker.getCurrentMinute());
           c.set(Calendar.SECOND,0);
           c.set(Calendar.MILLISECOND,0);

           /* 指定闹钟设置时间到时要运行 CallAlarm.class */
           Intent intent = new Intent(example9.this,
                                      example9 2.class);
           PendingIntent sender = PendingIntent.getBroadcast(
                                    example9.this,1, intent, 0);
           /* setRepeating()可让闹钟重复运行 */
           AlarmManager am;
           am = (AlarmManager)getSystemService(ALARM_SERVICE);
           am.setRepeating(AlarmManager.RTC_WAKEUP,
                     c.getTimeInMillis(),times,sender);
           /* 更新显示的设置闹钟时间 */
           String tmpS=format(tPicker.getCurrentHour())+": "+
                     format(tPicker.getCurrentMinute());
           setTime2.setText("设置闹钟时间为"+tmpS+
                         "开始，重复间隔为"+times/1000+"秒");
           /* 以 Toast 提示设置已完成 */
           Toast.makeText(example9.this,"设置闹钟时间为"+tmpS+
                       "开始，重复间隔为"+times/1000+"秒",
                       Toast.LENGTH_SHORT).show();
         }
       })
      .setNegativeButton("取消",
        new DialogInterface.OnClickListener()
       {
         public void onClick(DialogInterface dialog, int which)
         {
         }
       }).create();
```

(5) 实现重复响起闹钟的设置 Button。具体代码如下所示：

```
/* 重复响起闹钟的设置 Button */
mButton3=(Button) findViewById(R.id.mButton3);
mButton3.setOnClickListener(new View.OnClickListener()
{
  public void onClick(View v)
  {
    /* 取得点击按钮时的时间作为 tPicker 的默认值 */
    c.setTimeInMillis(System.currentTimeMillis());
    tPicker.setCurrentHour(c.get(Calendar.HOUR_OF_DAY));
    tPicker.setCurrentMinute(c.get(Calendar.MINUTE));
    /* 跳出设置画面 di */
    di.show();
  }
});
```

(6) 设置重复响起闹钟的删除 Button，首先在 AlarmManager 中删除，然后以 Toast 提示已删除设置并更新显示的闹钟时间。具体代码如下所示：

```
  /* 重复响起闹钟的删除 Button */
  mButton4=(Button) findViewById(R.id.mButton4);
  mButton4.setOnClickListener(new View.OnClickListener()
  {
    public void onClick(View v)
    {
      Intent intent = new Intent(example9.this, example9_2.class);
      PendingIntent sender = PendingIntent.getBroadcast(
                          example9.this,1, intent, 0);
      /* 在 AlarmManager 中删除 */
      AlarmManager am;
      am = (AlarmManager)getSystemService(ALARM_SERVICE);
      am.cancel(sender);
      /* 以 Toast 提示已删除设置，并更新显示的闹钟时间 */
      Toast.makeText(example9.this,"闹钟时间解除",
                  Toast.LENGTH_SHORT).show();
      setTime2.setText("目前无设置");
    }
  });
}
```

(7) 使用方法 format(int x)来设置日期时间显示两位数的显示格式。具体代码如下所示：

```
  /* 日期时间显示两位数的方法 */
  private String format(int x)
  {
    String s=""+x;
    if(s.length()==1) s="0"+s;
    return s;
  }
}
```

2. 文件 example9_1.java

文件 example9_1.java 用于实现实际跳出闹铃 Dialog 的 Activity，首先通过 AlertDialog.Builder (example9_1.this)实现弹出闹钟警示框，然后通过 onClick(DialogInterface dialog, int whichButton)关闭 Activity。具体代码如下所示：

```
/* 实际跳出闹铃 Dialog 的 Activity */
public class example9_1 extends Activity
{
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    /* 跳出的闹铃警示  */
    new AlertDialog.Builder(example9_1.this)
        .setIcon(R.drawable.clock)
        .setTitle("闹钟响了!!")
        .setMessage("赶快起床吧!!!")
        .setPositiveButton("关掉它",
         new DialogInterface.OnClickListener()
        {
          public void onClick(DialogInterface dialog, int whichButton)
          {
            /* 关闭 Activity */
            example9_1.this.finish();
          }
        })
        .show();
  }
}
```

3. 文件 example9_2.java

在文件 example9_2.java 中调用了闹钟 Alert 的 Receiver 并创建 Intent，用于调用 AlarmAlert.class。具体代码如下所示：

```
/* 调用闹钟 Alert 的 Receiver */
public class example9_2 extends BroadcastReceiver
{
  @Override
  public void onReceive(Context context, Intent intent)
  {
    /* create Intent，调用 AlarmAlert.class */
    Intent i = new Intent(context, example9_1.class);

    Bundle bundleRet = new Bundle();
    bundleRet.putString("STR_CALLER", "");
    i.putExtras(bundleRet);
    i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    context.startActivity(i);
  }
}
```

在文件 AndroidManifest.xml 中添加对 CallAlarm 的 receiver 设置。具体代码如下所示：

```
<!—注册 receiver CallAlarm -->
<receiver android:name=".example9_2" android:process=":remote" />
<activity android:name=".example9_1" ndroid:label="@string/app_name">
</activity>
```

至此整个任务完成。执行后的初始效果如图 11-17 所示，单击第一个“设置”按钮后在弹出的界面中可以设置闹钟时间，响一次的闹钟设置如图 11-18 所示。单击第二个按钮可以设置重复响起的时间如图 11-19 所示，闹钟响起后的界面如图 11-20 所示。



图 11-17　初始效果

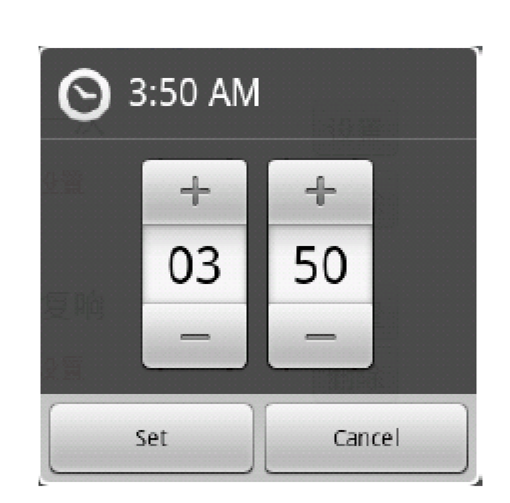


图 11-18　响一次的设置界面



图 11-19　重复响铃的设置界面



图 11-20　闹钟响起后的界面

11.9　黑名单拒绝你没商量

题　目	目　的	源码路径
题目 8	在 Android 中实现黑名单用户来电静音	“光盘:\daima\11\example10”文件夹

11.9.1　我的想法

几乎每个手机中都有黑名单功能，被列入黑名单的用户不能打进电话和发进短信。我决定先添加一个 EditText，用户可以在里面输入黑名单用户的号码。当此号码来电时，系统会自动切换成静音模式。当对方挂机后，再自动转换为正常模式，并使用 Toast 实现提示。铃声转换模式功能是通过 setRingerMode 实现的，正常模式是 RINGER_MODE_NORMAL，静音模式是 RINGER_MODE_SILENT，震动模式是 RINGER_MODE_VIBRATE。

11.9.2　具体实现

编写的主程序文件是 example10.java，其具体实现流程如下。

(1)　设置 PhoneCallListener 对象 phoneListener，用 TelephonyManager 抓取 Telephony Severice，然后设置 Listen Call，并查找 TextView EditText 的数据。具体代码如下所示：

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  /*设置 PhoneCallListener*/
  mPhoneCallListener phoneListener=new mPhoneCallListener();
  /*用 TelephonyManager 抓取 Telephony Severice*/
  TelephonyManager telMgr = (TelephonyManager)getSystemService(
      TELEPHONY_SERVICE);
  /*设置 Listen Call*/
  telMgr.listen(phoneListener, mPhoneCallListener.
      LISTEN_CALL_STATE);

  /*查找 TextView  EditText*/
  mTextView01 = (TextView)findViewById(R.id.myTextView1);
  mTextView03 = (TextView)findViewById(R.id.myTextView3);
  mEditText1 = (EditText)findViewById(R.id.myEditText1);

}
```

(2)　判断 PhoneStateListener 当前状态，获取手机待机状态后设置手机为待机时响铃正常；如果获取手机状态为通话中则显示对应信息，如果获取手机状态为来电则显示来电信息；然后判断输入电话是否一致，如果一样时用静音。具体代码如下所示：

```
/*  判断 PhoneStateListener 当前状态*/
public class mPhoneCallListener extends PhoneStateListener
{
  @Override
  public void onCallStateChanged(int state, String incomingNumber)
  {
    // TODO Auto-generated method stub
    switch(state)
    {
      /*  获取手机待机状态*/
      case TelephonyManager.CALL_STATE_IDLE:
        mTextView01.setText(R.string.str_CALL_STATE_IDLE);

        try
        {
          AudioManager audioManager = (AudioManager)
                       getSystemService(Context.AUDIO_SERVICE);
```

```
          if (audioManager != null)
          {
            /*设置手机为待机时响铃正常*/
            audioManager.setRingerMode(AudioManager.
                RINGER_MODE_NORMAL);
            audioManager.getStreamVolume(
                AudioManager.STREAM_RING);
          }
        }
        catch(Exception e)
        {
         mTextView01.setText(e.toString());
         e.printStackTrace();
        }
        break;

      /*  获取手机状态为通话中 */
      case TelephonyManager.CALL_STATE_OFFHOOK:
        mTextView01.setText(R.string.str_CALL_STATE_OFFHOOK);
        break;

      /*  获取手机状态为来电 */
      case TelephonyManager.CALL_STATE_RINGING:
        /*显示来电信息膭*/
        mTextView01.setText(
            getResources().getText(R.string.str CALL STATE RINGING)+
          incomingNumber);

        /*  判断输入电话是否一致，一样时用静音玻*/
        if(incomingNumber.equals(mTextView03.getText().toString()))
        {
        try
        {
          AudioManager audioManager = (AudioManager)
                        getSystemService(Context.AUDIO_SERVICE);
        if (audioManager != null)
          {
          /*设置响铃为静音玻*/
          audioManager.setRingerMode(AudioManager.
                RINGER_MODE_SILENT);
          audioManager.getStreamVolume(
                AudioManager.STREAM_RING);
          Toast.makeText(example10.this, getString(R.string.str_msg)
              ,Toast.LENGTH_SHORT).show();
          }
        }
        catch(Exception e)
        {
         mTextView01.setText(e.toString());
         e.printStackTrace();
        break;
        }
```

```
      }
    }

    super.onCallStateChanged(state, incomingNumber);
    mEditText1.setOnKeyListener(new EditText.OnKeyListener()
    {
```

(3)　定义 onKey(View v, int keyCode, KeyEvent event)，用于设置 EditText 的输入数据显示在 TextView。具体代码如下所示：

```
      @Override
      public boolean onKey(View v, int keyCode, KeyEvent event)
      {
        // TODO Auto-generated method stub
        /*设置 EditText 的输入数据显示在 TextView*/
        mTextView03.setText(mEditText1.getText());
        return false;
      }
```

至此整个任务完成。执行后的初始效果如图 11-21 所示，在输入框中可以输入黑名单号码，如图 11-22 所示。这样当此号码来电时会显示静音模式，如图 11-23 所示。



图 11-21　初始效果



图 11-22　黑名单号码



图 11-23　来电静音

11.10　指定时间置换桌面背景

题　目	目　的	源码路径
题目 9	在指定时间置换桌面背景	“光盘:\daima\11\example11”文件夹

11.10.1　我的想法

在移动手机设备中，为用户提供了在不同时间显示不同屏幕背景照片的功能。我发现此功能和前面的闹钟实例类似，实际上 AlarmManage 并不是只能用作闹钟，它还可以自行设置在什么时间运行什么样的动作。在正式开始之前我预先准备了 7 张素材图片供选择，图片放在了“res\drawable”目录下。

11.10.2　具体实现

编写的主程序文件是 example11.java、MyReceiver.java、ChangeBgImage.java 和 DailyBgDB.java。

1. 文件 example11.java

(1) 分别声明自定义的数据库变量 DailyBgDB，存放设置值的 map，存放图文件 id 的数组 bg 与存放图文件名称的数组 bgName。具体代码如下所示：

```
/* 声明自定义的数据库变量 DailyBgDB */
private DailyBgDB db;
/* 声明存放设置值的 Map */
private Map<Integer,Integer> map;
private LayoutInflater inflater;
private int tmpWhich=0;
/* 声明存放图文件 id 的数组 bg 与存放图文件名称的数组 bgName */
private final int[] bg =
{R.drawable.b01,R.drawable.b02,R.drawable.b03,R.drawable.b04,
R.drawable.b05,R.drawable.b06,R.drawable.b07};
private final String[] bgName =
{"b01.png","b02.png","b03.png","b04.png","b05.png","b06.png",
"b07.png"};
```

(2) 载入主布局文件 main.xml 并将数据库存放的设置值放入 map 中，然后初始化各个 TextView 对象。具体代码如下所示：

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  /* 载入 main.xml Layout */
  setContentView(R.layout.main);
  inflater=(LayoutInflater)getSystemService(
     Context.LAYOUT_INFLATER_SERVICE);

  /* 将数据库存放的设置值放入 map 中 */
```

```
initSettingData();
/* 初始化 TextView 对象 */
mySet1=(TextView) findViewById(R.id.mySet1);
mySet2=(TextView) findViewById(R.id.mySet2);
mySet3=(TextView) findViewById(R.id.mySet3);
mySet4=(TextView) findViewById(R.id.mySet4);
mySet5=(TextView) findViewById(R.id.mySet5);
mySet6=(TextView) findViewById(R.id.mySet6);
mySet7=(TextView) findViewById(R.id.mySet7);
```

(3) 根据获取的图像设置显示的图文件名称。具体代码如下所示：

```
/* 设置显示的图文件名称 */
if(!map.get(0).equals(99))
{
  mySet1.setText(bgName[map.get(0)]);
}
if(!map.get(1).equals(99))
{
  mySet2.setText(bgName[map.get(1)]);
}
if(!map.get(2).equals(99))
{
  mySet3.setText(bgName[map.get(2)]);
}
if(!map.get(3).equals(99))
{
  mySet4.setText(bgName[map.get(3)]);
}
if(!map.get(4).equals(99))
{
  mySet11.setText(bgName[map.get(4)]);
}
if(!map.get(5).equals(99))
{
  mySet11.setText(bgName[map.get(5)]);
}
if(!map.get(6).equals(99))
{
  mySet7.setText(bgName[map.get(6)]);
}
```

(4) 初始化各个 Button 对象，然后以 initButton()方法监听按钮事件。具体代码如下所示：

```
/* 初始化 Button 对象 */
setButton1=(Button) findViewById(R.id.setButton1);
setButton2=(Button) findViewById(R.id.setButton2);
setButton3=(Button) findViewById(R.id.setButton3);
setButton4=(Button) findViewById(R.id.setButton4);
setButton5=(Button) findViewById(R.id.setButton5);
setButton6=(Button) findViewById(R.id.setButton6);
setButton7=(Button) findViewById(R.id.setButton7);
/* 以 initButton()来设置 OnClickListener */
```

```
setButton1=initButton(setButton1,mySet1,0);
setButton2=initButton(setButton2,mySet2,1);
setButton3=initButton(setButton3,mySet3,2);
setButton4=initButton(setButton4,mySet4,3);
setButton5=initButton(setButton5,mySet5,4);
setButton6=initButton(setButton6,mySet6,5);
setButton7=initButton(setButton7,mySet7,6);
```

(5) 设置启动服务的 Button，通过 setOnClickListener 实现单击事件的监听。具体实现流程如下。

第 1 步：取得服务启动后一天的 0 点 0 分 0 秒的 millsTime。

第 2 步：重复运行的间隔时间，并将更换桌布的排程添加到 AlarmManager 中。

第 3 步：通过 setRepeating()让排程处理重复运行，并通过 Toast 提示已启动。

第 4 步：启动后马上先运行一次换桌布的程序来更换今天的桌布。

具体代码如下所示：

```
/* 设置启动服务的 Button */
mButton1=(Button)findViewById(R.id.myButton1);
mButton1.setOnClickListener(new View.OnClickListener()
{
  public void onClick(View v)
  {
    /* 取得服务启动后一天的 0 点 0 分 0 秒的 millsTime */
    Calendar calendar=Calendar.getInstance();
    calendar.add(Calendar.DATE,1);
    calendar.set(Calendar.HOUR_OF_DAY,0);
    calendar.set(Calendar.MINUTE,0);
    calendar.set(Calendar.SECOND,0);
    calendar.set(Calendar.MILLISECOND,0);
    long startTime=calendar.getTimeInMillis();
    /* 重复运行的间隔时间 */
    long repeatTime=24*60*60*1000;
    /* 将更换桌布的排程添加到 AlarmManager 中 */
    Intent intent = new Intent(example11.this,MyReceiver.class);
    PendingIntent sender = PendingIntent.getBroadcast(
                          example11.this, 0, intent, 0);
    AlarmManager am = (AlarmManager)getSystemService(
      ALARM_SERVICE);
    /* setRepeating()可让排程重复运行
       startTime 为开始运行时间
       repeatTime 为重复运行间隔
       AlarmManager.RTC 可使服务休眠时仍然会运行 */
    am.setRepeating(AlarmManager.RTC,startTime,repeatTime,
                 sender);
    /* 以 Toast 提示已启动 */
    Toast.makeText(example11.this,"服务已启动",Toast.LENGTH_SHORT)
      .show();
    /* 启动后马上先运行一次换桌布的程序以更换今天的桌布 */
    Intent i = new Intent(example11.this,ChangeBgImage.class);
    startActivity(i);
  }
});
```

(6)　设置终止服务的 Button，定义 onClick(View v)实现对单击的监听。首先定义 Intent 对象 intent；然后在 AlarmManager 中删除调度；最后通过 Toast 提示已终止。具体代码如下所示：

```
  /* 设置终止服务的 Button */
  mButton2=(Button) findViewById(R.id.myButton2);
  mButton2.setOnClickListener(new View.OnClickListener()
  {
    public void onClick(View v)
    {
      Intent intent = new Intent(example11.this,MyReceiver.class);
      PendingIntent sender = PendingIntent.getBroadcast(
                          example11.this, 0, intent, 0);
      /* 由 AlarmManager 中删除调度 */
      AlarmManager am = (AlarmManager)getSystemService(
        ALARM_SERVICE);
      am.cancel(sender);
      /* 以 Toast 提示已终止 */
      Toast.makeText(example11.this,"服务已终止",Toast.LENGTH_SHORT)
        .show();
    }
  });
}
```

(7)　定义方法 initSettingData()，用于从数据库中取得设置值的方法。具体代码如下所示：

```
/* 由数据库中取得设置值的方法 */
private void initSettingData()
{
  map=new LinkedHashMap<Integer,Integer>();
  db=new DailyBgDB(example11.this);
  Cursor cur=db.select();
  while(cur.moveToNext()){
    map.put(cur.getInt(0),cur.getInt(1));
  }
  cur.close();
  db.close();
}
```

(8)　设置 Button 的 OnClickListener 方法，首先设置单击 Button 后跳出的选择图片的 Dialog，然后设置预览画面的文件名与 ImageView，最后改变画面显示的设置图文件名并将更改的设置存入数据库。具体代码如下所示：

```
/* 设置 Button 的 OnClickListener 方法 */
private Button initButton(Button b,final TextView t,final int id)
{
  b.setOnClickListener(new View.OnClickListener()
  {
    public void onClick(View v)
    {
      /* 设置点击 Button 后跳出的选择图片的 Dialog */
      new AlertDialog.Builder(example11.this)
```

```
        .setIcon(R.drawable.pic icon)
        .setTitle("请选择图片！")
        .setSingleChoiceItems(bgName,map.get(id),
         new DialogInterface.OnClickListener()
         {
           public void onClick(DialogInterface dialog,int which)
           {
             tmpWhich=which;
             /* 选择图片后跳出预览图文件的窗口 */
             View view=inflater.inflate(R.layout.preview, null);
             TextView message=(TextView) view.findViewById(
               R.id.bgName);
             /* 设置预览画面的文件名与 ImageView */
             message.setText(bgName[which]);
             ImageView mView01 = (ImageView)view.findViewById(
               R.id.bgImage);
             mView01.setImageResource(bg[which]);

             Toast toast=Toast.makeText(example11.this,"",
               Toast.LENGTH_SHORT);
             toast.setView(view);
             toast.show();
           }
         })
        .setPositiveButton("确定",
         new DialogInterface.OnClickListener()
        {
          public void onClick(DialogInterface dialog,int which1)
          {
            /* 改变画面显示的设置图文件名 */
            t.setText(bgName[tmpWhich]);
            /* 改变 map 里的值 */
            map.put(id,tmpWhich);
            /* 将更改的设置存入数据库 */
            saveData(id,tmpWhich);
          }
        })
        .setNegativeButton("取消",
         new DialogInterface.OnClickListener()
        {
          public void onClick(DialogInterface dialog,int which)
          {
          }
        }).show();
    }
  });
  return b;
}
```

(9) 定义 saveData(int id,int value)，用于存储设置值至 DB 的方法。具体代码如下所示：

```
/* 存储设置值至 DB 的方法 */
private void saveData(int id,int value)
```

```
  {
    db=new DailyBgDB(example11.this);
    db.update(id,value);
    db.close();
  }
}
```

2. 文件 MyReceiver.java

文件 MyReceiver.java 的功能是运行更换桌面背景的 Receiver。主要代码如下所示：

```
/* 运行更换桌面背景的 Receiver */
public class MyReceiver extends BroadcastReceiver
{
  @Override
  public void onReceive(Context context, Intent intent)
  {
    /* create Intent，调用 ChangeBgImage.class */
    Intent i = new Intent(context, ChangeBgImage.class);

    Bundle bundleRet = new Bundle();
    bundleRet.putString("STR_CALLER", "");
    i.putExtras(bundleRet);
    i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    context.startActivity(i);
  }
}
```

3. 文件 ChangeBgImage.java

文件 ChangeBgImage.java 的具体实现流程如下所示。

(1) 运行更换桌面背景的 Activity，并声明存放图文件 id 的数组 bg。具体代码如下所示：

```
/* 实际运行更换桌面背景的 Activity */
public class ChangeBgImage extends Activity
{
  /* 声明存放图文件 id 的数组 bg */
  private static final int[] bg =
    {R.drawable.b01,R.drawable.b02,R.drawable.b03,R.drawable.b04,
    R.drawable.b05,R.drawable.b06,R.drawable.b07};
```

(2) 载入布局文件 progress.xml 并获取今天是星期几，然后从数据库中取得今天应该换哪张背景，如果 DailyBg==99 代表没设置则不运行。具体代码如下所示：

```
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    /* 载入 progress.xml Layout */
    setContentView(R.layout.progress);
    /* 取得今天是星期几 */
    Calendar ca=Calendar.getInstance();
    int dayOfWeek=ca.get(Calendar.DAY_OF_WEEK)-1;
```

```
    /* 从数据库中取得今天应该换哪一张背景 */
    int DailyBg=0;
    String selection = "DailyId=?";
    String[] selectionArgs = new String[]{""+dayOfWeek};
    DailyBgDB db=new DailyBgDB(ChangeBgImage.this);
    Cursor cur=db.select(selection,selectionArgs);
    while(cur.moveToNext())
    {
      DailyBg=cur.getInt(0);
    }
    cur.close();
    db.close();
    /* 如果 DailyBg==99 代表没设置，所以不运行 */
    if(DailyBg!=99)
    {
      Bitmap bmp=BitmapFactory.decodeResource
      (getResources(), bg[DailyBg]);
      try
      {
        super.setWallpaper(bmp);
      }
      catch (IOException e)
      {
        e.printStackTrace();
      }
    }
    finish();
  }
}
```

4. 文件 DailyBgDB.java

文件 DailyBgDB.java 的具体实现流程如下。

(1) 定义构造器 DailyBgDB(Context context)。具体代码如下所示：

```
/* 构造器 */
public DailyBgDB(Context context)
{
  super(context, DATABASE_NAME, null, DATABASE_VERSION);
  sdb= this.getWritableDatabase();
}
```

(2) 实现数据库处理，如果 Table 不存在就创建 table 并存入初始的数据到 DB，具体代码如下：

```
@Override
public void onCreate(SQLiteDatabase db)
{
  /* Table 不存在就创建 table */
  String sql = "CREATE TABLE IF NOT EXISTS "+TABLE_NAME+"("+FIELD1
      +" INTEGER primary key, "+FIELD2+" INTEGER)";
  db.execSQL(sql);
```

```
    /* 存入初始的数据到 DB */
    sdb=db;
    insert(0,99);
    insert(1,99);
    insert(2,99);
    insert(3,99);
    insert(4,99);
    insert(5,99);
    insert(6,99);
  }
  @Override
  public void onUpgrade(SQLiteDatabase db,int oldVersion,
                     int newVersion)
  {
  }
  public Cursor select()
  {
    Cursor cursor=sdb.query(TABLE_NAME,null,null,
                        null,null,null,null);
    return cursor;
  }
```

(3)　定义方法 select(String selection,String[] selectionArgs)，当 select 有 where 条件时要用此方法，用于检索数据。具体代码如下所示：

```
  /* select 时有 where 条件要用此方法 */
  public Cursor select(String selection,String[] selectionArgs)
  {
    String[] columns = new String[] { FIELD2 };
    Cursor cursor=sdb.query(TABLE_NAME,columns,selection,
                        selectionArgs,null,null,null);
    return cursor;
  }
```

(4)　定义方法 insert(int value1,int value2)，用于将添加的值放入 ContentValues。具体代码如下所示：

```
  public long insert(int value1,int value2)
  {
    /* 将添加的值放入 ContentValues */
    ContentValues cv = new ContentValues();
    cv.put(FIELD1, value1);
    cv.put(FIELD2, value2);
    long row = sdb.insert(TABLE_NAME, null, cv);
    return row;
  }
```

(5)　定义方法 delete(int id)，用于删除设置。具体代码如下所示：

```
  public void delete(int id)
  {
    String where = FIELD1 + " = ?";
```

```
    String[] whereValue ={ Integer.toString(id) };
    sdb.delete(TABLE_NAME, where, whereValue);
  }
```

(6) 定义 update(int id, int value)方法，用于修改设置。具体代码如下所示：

```
  public void update(int id, int value)
  {
    String where = FIELD1 + " = ?";
    String[] whereValue ={ Integer.toString(id) };
    /* 将修改的值放入 ContentValues */
    ContentValues cv = new ContentValues();
    cv.put(FIELD2, value);
    sdb.update(TABLE_NAME, cv, where, whereValue);
  }
}
```

在文件 AndroidManifest.xml 中添加 MyReceiver 的 receiver 设置，并添加背景图像权限 android.permission.SET_WALLPAPER。具体代码如下所示：

```
    <receiver android:name=".MyReceiver" android:process=":remote"/>
    <!-- 设定 SET_WALLPAPER 权限 -->
    <uses-permission android:name="android.permission.SET_WALLPAPER" />
```

至此整个任务完成。执行后的初始效果如图 11-24 所示，选择一个星期几，单击后面的“设置”按钮可以在弹出的界面中设置当天的背景图片，如图 11-25 所示。



图 11-24 初始效果



图 11-25 设置背景图片

11.11 设计开机显示程序

题 目	目 的	源码路径
题目 10	设计一个开机显示程序	“光盘:\daima\11\example10”文件夹

11.11.1　我的想法

在很多移动手机设备中，当开机后一般都会显示一个特定的开机界面。我也知道前面讲解的 Activity、Service 和 Broadcast Reseiver 都是在开机之后运行的。实际上在此时开机事件会发送出一个 Android.intent.action.BOOT_COMPLETED 广播信息，只要接收到此 Action 就能在 Reseiver 中打开自己的程序。我决定设计一个 Activity 和一个 Broadcast Reseiver 类，只要此程序运行一次，以后一开机就会自动运行这个程序。

11.11.2　具体实现

编写的主程序文件是 example13.java 和 StartupIntentReceiver.java，其具体实现流程如下。

1. 文件 example13.java

文件 example13.java 的功能是定义主程序 Activity，本程序只要运行一次就会在日后开机时自动运行，并以欢迎的 TextView 文字作为提示文本。具体代码如下所示：

```
public class example13 extends Activity
{
  /* 本程序只要运行一次，就会在日后开机时自动运行 */
  private TextView mTextView01;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    /* 为了快速示意，程序仅以欢迎的 TextView 文字作为展示 */
    mTextView01 = (TextView)findViewById(R.id.myTextView1);
    mTextView01.setText(R.string.str_welcome);
  }
}
```

2. 文件 StartupIntentReceiver.java

在文件 StartupIntentReceiver.java 中添加了一个继承自 BroadcastReceiver 类的 StartupIntentReceiver 类，在其内部覆盖了 onReceive()方法，此方法会接收来自系统的广播。onReceive()方法的唯一任务是把自己唤醒，所以在传入 Intent 参数中的第二个参数是指定 Activity 的 class，最后以 start-Activity()方法打开并运行程序。具体代码如下所示：

```
/* 捕捉 android.intent.action.BOOT_COMPLETED 的 Receiver 类 */
public class StartupIntentReceiver extends BroadcastReceiver
{
  @Override
  public void onReceive(Context context, Intent intent)
  {
    // TODO Auto-generated method stub
```

```
    /* 当收到 Receiver 时，指定打开此程序(EX06_111.class) */
    Intent mBootIntent = new Intent(context, example13.class);

    /* 设置 Intent 打开为 FLAG_ACTIVITY_NEW_TASK */
    mBootIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

    /* 将 Intent 以 startActivity 传送给操作系统 */
    context.startActivity(mBootIntent);
  }
}
```

至此整个任务完成，执行后将会显示预先设置的开机欢迎语，执行效果如图 11-26 所示。

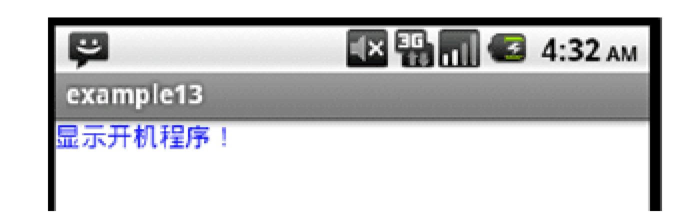


图 11-26　执行效果



第 12 章　第三关：江湖笑

在移动手机应用中，娱乐和多媒体是一个重要的构成模块，主要包含了屏保、图片、MP3 播放、影片播放和相机照片等。在本节的内容中，将通过几个典型实例的实现过程来详细介绍 Android 中娱乐和多媒体编程的基本知识。

12.1　驻 足 江 湖

Android 应用是为消费者服务的，好的用户体验必能带来可观的经济效益。作为 Android 江湖中的一名程序员，我们要尽量开发出界面绚丽的应用项目。在我的概念中，流畅的操作、漂亮的画面、丰富的系统功能能够带给用户耳目一新的感受。我一直追求最完美的品质、最优质的服务，将风格唯美的画面、丰富多彩的应用充分融合在项目中，获取广大用户们的好评。武林大会的第三关——驻足江湖，题目都和娱乐有关，目的是鼓励开发人员开发出绚丽多彩的娱乐和多媒体项目。

12.2　绘制几何图形

题　目	目　的	源码路径
题目 1	在 Android 中绘制几何图形	“光盘:\daima\12\example2”文件夹

12.2.1　我的想法

当前手机应用中，很多游戏程序中需要有绘制几何图形的功能。这些功能实现起来很简单，可以通过 Android.graphics 类来绘制 2D 向量图。它提供了很多在手机上绘制图形的类和方法。例如 Canvas 相当于一个图纸，所有的图形都能在它上面绘制并显示出来；而 Paint 则像铅笔，可以设置为不同的颜色，从而绘制不同颜色的图形。

12.2.2 具体实现

编写的主程序文件是 example2.java，其具体实现代码如下。

(1) 自定义继承 View 的 MyView，分别设置背景和消除锯齿，然后分别绘制空心圆形、空心正方形、空心长方形、空心椭圆形、空心三角形和空心梯形。具体代码如下所示：

```
/* 自定义继承 View 的 MyView */
private class MyView extends View
{
  public MyView(Context context)
  {
    super(context);
  }
  /* 覆盖 onDraw() */
  @Override
  protected void onDraw(Canvas canvas)
  {
    super.onDraw(canvas);
    /* 设置背景为白色 */
    canvas.drawColor(Color.WHITE);
    Paint paint = new Paint();
    /* 去锯齿 */
    paint.setAntiAlias(true);
    /* 设置 paint 的颜色 */
    paint.setColor(Color.RED);
    /* 设置 paint 的 style 为 STROKE：空心的 */
    paint.setStyle(Paint.Style.STROKE);
    /* 设置 paint 的外框宽度 */
    paint.setStrokeWidth(3);

    /* 画一个空心圆形 */
    canvas.drawCircle(40,40,30, paint);
    /* 画一个空心正方形 */
    canvas.drawRect(10,90,70,150,paint);
    /* 画一个空心长方形 */
    canvas.drawRect(10,170,70,200,paint);
    /* 画一个空心椭圆形 */
    RectF re=new RectF(10,220,70,250);
    canvas.drawOval(re, paint);
    /* 画一个空心三角形 */
    Path path = new Path();
    path.moveTo(10,330);
    path.lineTo(70,330);
    path.lineTo(40,270);
    path.close();
    canvas.drawPath(path, paint);
    /* 画一个空心梯形 */
    Path path1 = new Path();
    path1.moveTo(10,410);
    path1.lineTo(70,410);
```

```
    path1.lineTo(55,350);
    path1.lineTo(25,350);
    path1.close();
    canvas.drawPath(path1, paint);
```

(2) 设置实心样式和颜色，然后分别绘制实心圆形、实心正方形、实心长方形、实心椭圆形、实心三角形和实心梯形。具体代码如下所示：

```
    /* 设置paint的style为FILL：实心 */
    paint.setStyle(Paint.Style.FILL);
    /* 设置paint的颜色 */
    paint.setColor(Color.BLUE);
    /* 画一个实心圆 */
    canvas.drawCircle(120, 40, 30, paint);
    /* 画一个实心正方形 */
    canvas.drawRect(90,90,150,150,paint);
    /* 画一个实心长方形 */
    canvas.drawRect(90,170,150,200,paint);
    /* 画一个实心椭圆形 */
    RectF re2=new RectF(90,220,150,250);
    canvas.drawOval(re2, paint);
    /* 画一个实心三角形 */
    Path path2 = new Path();
    path2.moveTo(90,330);
    path2.lineTo(150,330);
    path2.lineTo(120,270);
    path2.close();
    canvas.drawPath(path2, paint);
    /* 画一个实心梯形 */
    Path path3 = new Path();
    path3.moveTo(90,410);
    path3.lineTo(150,410);
    path3.lineTo(135,350);
    path3.lineTo(105,350);
    path3.close();
    canvas.drawPath(path3, paint);
```

(3) 设置渐变样式和颜色，然后分别绘制渐变圆形、渐变正方形、渐变长方形、渐变椭圆形、渐变三角形和渐变梯形。具体代码如下所示：

```
    /* 设置渐变色 */
    Shader mShader=new LinearGradient(0, 0,100,100,
        new int[]{Color.RED, Color.GREEN,Color.BLUE,Color.YELLOW},
        null, Shader.TileMode.REPEAT);
    paint.setShader(mShader);
    /* 画一个渐变色的圆形 */
    canvas.drawCircle(200,40, 30, paint);
    /* 画一个渐变色的正方形 */
    canvas.drawRect(170,90,230,150,paint);
    /* 画一个渐变色的长方形 */
    canvas.drawRect(170,170,230,200,paint);
    /* 画一个渐变色的椭圆形 */
    RectF re3=new RectF(170,220,230,250);
    canvas.drawOval(re3, paint);
```

```
    /* 画一个渐变色的三角形 */
    Path path4 = new Path();
    path4.moveTo(170,330);
    path4.lineTo(230,330);
    path4.lineTo(200,270);
    path4.close();
    canvas.drawPath(path4, paint);
    /* 画一个渐变色的梯形 */
    Path path5 = new Path();
    path5.moveTo(170,410);
    path5.lineTo(230,410);
    path5.lineTo(215,350);
    path5.lineTo(185,350);
    path5.close();
    canvas.drawPath(path5, paint);
```

(4) 通过 canvas.drawText 实现写字功能。具体代码如下所示：

```
    /* 写字 */
    paint.setTextSize(24);
    canvas.drawText(getResources().getString(R.string.str_text1),
                240,50,paint);
    canvas.drawText(getResources().getString(R.string.str_text2),
                240,120,paint);
    canvas.drawText(getResources().getString(R.string.str_text3),
                240,190,paint);
    canvas.drawText(getResources().getString(R.string.str_text4),
                240,250,paint);
    canvas.drawText(getResources().getString(R.string.str_text5),
                240,320,paint);
    canvas.drawText(getResources().getString(R.string.str_text6),
                240,390,paint);
   }
  }
}
```

至此整个展示结束。执行后将会在屏幕内显示不同的图形，并在图形后面显示对应的文字描述，执行效果如图 12-1 所示。

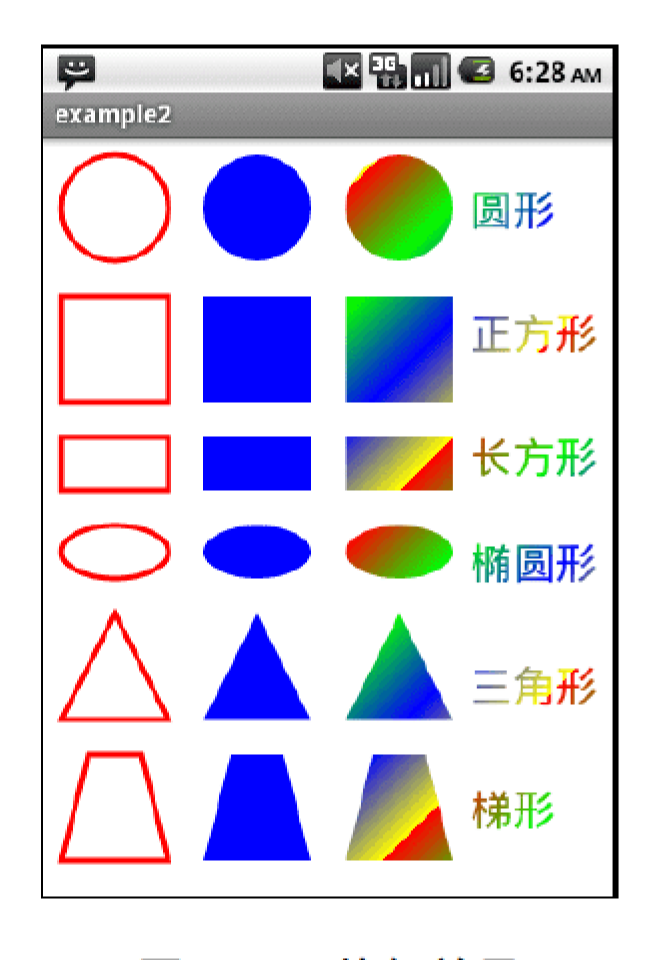


图 12-1　执行效果

12.3　屏保程序的魅力

题　目	目　的	源码路径
题目 2	在 Android 中实现屏保功能	“光盘:\daima\12\example3”文件夹

12.3.1　我的想法

在当前手机应用中屏幕程序很常见，但是对我来说有两个难点很难解决。

(1)　控制和判断用户静止未触动手机键盘或屏幕的时间及其事件。

(2)　通过动态全屏幕淡入、淡出和图片交换效果。

在《Android SDK 4.0 密录》中我找到了答案，原来上述难点都是通过线程实现的，它以时间戳记的方式，判断距离上一次单击键盘或屏幕的时间，并计算两次的间隔。当超过了设置的时间后会进入屏保程序，本实例设置的时间间隔为 5 秒。

12.3.2　具体实现

编写的主程序文件是 example3.java，其具体实现代码如下。

(1)　分别定义控制用户静止与否的 Counter，控制 FadeIn 与 Fade Out 的 Counter，控制循序替换背景图 ID 的 Counter。具体代码如下所示：

```
/* 控制 User 静止与否的 Counter */
private int intCounter1, intCounter2;
/* 控制 FadeIn 与 Fade Out 的 Counter */
private int intCounter3, intCounter4;
/* 控制循序替换背景图 ID 的 Counter  */
private int intDrawable=0;
```

(2)　设置 timePeriod，设置当静止超过 n 秒将自动进入屏幕保护。具体代码如下所示：

```
/* 上一次 User 有动作的 Time Stamp */
private Date lastUpdateTime;
/* 计算 User 共几秒没有动作 */
private long timePeriod;
/* 静止超过 n 秒将自动进入屏幕保护 */
private float fHoldStillSecond = (float) 5;
private boolean bIfRunScreenSaver;
private boolean bFadeFlagOut, bFadeFlagIn = false;
private long intervalScreenSaver = 1000;
private long intervalKeypadeSaver = 1000;
private long intervalFade = 100;
private int screenWidth, screenHeight;
```

(3)　设置每 5 秒置换图片，用 Screen Saver 保存需要用到的背景图。具体代码如下所示：

```
/* 每 n 秒置换图片 */
private int intSecondsToChange = 5;
```

```
/* 设置 Screen Saver 需要用到的背景图 */
private static int[] screenDrawable = new int[]
{
  R.drawable.screen1,
  R.drawable.screen2,
  R.drawable.screen3,
  R.drawable.screen4,
  R.drawable.screen5
};
```

(4) 设置在 setContentView 之前调用全屏幕显示，通过 lastUpdateTime 初始取得 User 触碰手机的时间，并用 recoverOriginalLayout()来初始化 Layout 上的 Widget 可见性。具体代码如下所示：

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);

  /* 必须在 setContentView 之前调用全屏幕显示 */
  requestWindowFeature(Window.FEATURE_NO_TITLE);
  getWindow().setFlags
  (
    WindowManager.LayoutParams.FLAG_FULLSCREEN,
    WindowManager.LayoutParams.FLAG FULLSCREEN
  );
  setContentView(R.layout.main);

  /* onCreate all Widget */
  mTextView01 = (TextView)findViewById(R.id.myTextView1);
  mImageView01 = (ImageView)findViewById(R.id.myImageView1);
  mEditText01 = (EditText)findViewById(R.id.myEditText1);

  /* 初始取得 User 触碰手机的时间 */
  lastUpdateTime = new Date(System.currentTimeMillis());

  /* 初始化 Layout 上的 Widget 可见性 */
  recoverOriginalLayout();
}
```

(5) 设置 Menu 群组 ID，然后通过 menu.add 创建具有 SubMenu 的 Menu，最后创建退出 Menu。具体代码如下所示：

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
  // TODO Auto-generated method stub

  /* menu 群组 ID */
  int idGroup1 = 0;
```

```
  /* The order position of the item */
  int orderMenuItem1 = Menu.NONE;
  int orderMenuItem2 = Menu.NONE+1;
  /* 创建具有 SubMenu 的 menu */
  menu.add
  (
    idGroup1, MENU_ABOUT, orderMenuItem1, R.string.app_about
  );
  /* 创建退出 Menu */
  menu.add(idGroup1, MENU_EXIT, orderMenuItem2, R.string.str_exit);
  menu.setGroupCheckable(idGroup1, true, true);

  return super.onCreateOptionsMenu(menu);
}
```

(6) 根据用户选择的 Menu，显示对应的 AlertDialog 提示框。具体代码如下所示：

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
  // TODO Auto-generated method stub
  switch(item.getItemId())
  {
    case (MENU_ABOUT):
      new AlertDialog.Builder
      (
        example3.this
      ).setTitle(R.string.app_about).setIcon
      (
        R.drawable.hippo
      ).setMessage
      (
        R.string.app_about_msg
      ).setPositiveButton(R.string.str ok,
      new DialogInterface.OnClickListener()
      {
        public void onClick
        (DialogInterface dialoginterface, int i)
        {
        }
      }).show();
      break;
    case (MENU_EXIT):
      /* 离开程序 */
      finish();
      break;
  }
  return super.onOptionsItemSelected(item);
}
```

(7) 用 mTasks01 监控 User 没有动作的运行线程，通过 timePeriod 计算 User 静止不动作的时间间距，如果静止不动超过设置的 5 秒则运行对应的线程。具体代码如下所示：

```
/* 监控 User 没有动作的运行线程 */
private Runnable mTasks01 = new Runnable()
{
  public void run()
  {
    intCounter1++;
    Date timeNow = new Date(System.currentTimeMillis());
    /* 计算 User 静止不动作的时间间距 */
    timePeriod =
    (long)timeNow.getTime() - (long)lastUpdateTime.getTime();

    float timePeriodSecond = ((float)timePeriod/1000);

    /* 如果超过时间静止不动 */
    if(timePeriodSecond>fHoldStillSecond)
    {
      /* 静止超过时间第一次的标记 */
      if(bIfRunScreenSaver==false)
      {
        /* 启动运行线程 2 */
        mHandler02.postDelayed(mTasks02, intervalScreenSaver);

        /* Fade Out*/
        if(intCounter1%(intSecondsToChange)==0)
        {
          bFadeFlagOut=true;
          mHandler03.postDelayed(mTasks03, intervalFade);
        }
        else
        {
          /* 在 Fade Out 后立即 Fade In */
          if(bFadeFlagOut==true)
          {
            bFadeFlagIn=true;
            mHandler04.postDelayed(mTasks04, intervalFade);
          }
          else
          {
            bFadeFlagIn=false;
            intCounter4 = 0;
            mHandler04.removeCallbacks(mTasks04);
          }
          intCounter3 = 0;
          bFadeFlagOut = false;
        }
        bIfRunScreenSaver = true;
      }
      else
      {
        /* screen saver 正在运行中 */
        /* Fade Out*/
        if(intCounter1%(intSecondsToChange)==0)
```

```
        {
          bFadeFlagOut=true;
          mHandler03.postDelayed(mTasks03, intervalFade);
        }
        else
        {
          /* 在 Fade Out 后立即 Fade In */
          if(bFadeFlagOut==true)
          {
            bFadeFlagIn=true;
            mHandler04.postDelayed(mTasks04, intervalFade);
          }
          else
          {
            bFadeFlagIn=false;
            intCounter4 = 0;
            mHandler04.removeCallbacks(mTasks04);
          }
          intCounter3 = 0;
          bFadeFlagOut=false;
        }
      }
    }
    else
    {
      /* 当 User 没有动作的间距未超过时间 */
      bIfRunScreenSaver = false;
      /* 恢复原来的 Layout Visible*/
      recoverOriginalLayout();
    }

    /* 以 LogCat 监看 User 静止不动的时间间距 */
    Log.i
    (
      "HIPPO",
      "Counter1:"+Integer.toString(intCounter1)+
      "/"+
      Float.toString(timePeriodSecond));
    /* 反复运行线程 1 */
    mHandler01.postDelayed(mTasks01, intervalKeypadeSaver);
  }
};
```

(8) 定义 mTasks02，设置每 1 秒运行一次屏保程序并隐藏原有 Layout 的 Widget，调用 ScreenSaver()加载图片，即轮换显示预设的 5 幅图片。具体代码如下所示：

```
/* Screen Saver Runnable */
private Runnable mTasks02 = new Runnable()
{
  public void run()
  {
    if(bIfRunScreenSaver==true)
```

```
        {
          intCounter2++;

          hideOriginalLayout();
          showScreenSaver();
          //Log.i("HIPPO", "Counter2:"+Integer.toString(intCounter2));
          mHandler02.postDelayed(mTasks02, intervalScreenSaver);
        }
        else
        {
          mHandler02.removeCallbacks(mTasks02);
        }
      }
    };
```

(9) 定义 mTasks03，通过 setAlpha 设置 ImageView 的透明度渐暗下去。具体代码如下所示：

```
/* Fade Out 特效 Runnable */
private Runnable mTasks03 = new Runnable()
{
  public void run()
  {
    if(bIfRunScreenSaver==true && bFadeFlagOut==true)
    {
      intCounter3++;

      /* 设置 ImageView 的透明度渐暗下去 */
      mImageView01.setAlpha(255-intCounter3*28);
      Log.i("HIPPO", "Fade out:"+Integer.toString(intCounter3));
      mHandler03.postDelayed(mTasks03, intervalFade);
    }
    else
    {
      mHandler03.removeCallbacks(mTasks03);
    }
  }
};
```

(10) 定义 mTasks04，通过 setAlpha 设置 ImageView 的透明度渐亮起来。具体代码如下所示：

```
/* Fade In 特效 Runnable */
private Runnable mTasks04 = new Runnable()
{
  public void run()
  {
    if(bIfRunScreenSaver==true && bFadeFlagIn==true)
    {
      intCounter4++;

      /* 设置 ImageView 的透明度渐亮起来 */
      mImageView01.setAlpha(intCounter4*28);

      mHandler04.postDelayed(mTasks04, intervalFade);
```

```
        Log.i("HIPPO", "Fade In:"+Integer.toString(intCounter4));
      }
      else
      {
        mHandler04.removeCallbacks(mTasks04);
      }
    }
  };
```

(11) 先定义 recoverOriginalLayout()方法，用于恢复原有的 Layout 可视性，然后定义 hideOriginalLayout()方法，用于隐藏原有应用程序中的布局配置组件。具体代码如下所示：

```
/* 恢复原有的 Layout 可视性 */
private void recoverOriginalLayout()
{
  mTextView01.setVisibility(View.VISIBLE);
  mEditText01.setVisibility(View.VISIBLE);
  mImageView01.setVisibility(View.GONE);
}
/* 隐藏原有应用程序里的布局配置组件 */
private void hideOriginalLayout()
{
  /* 将欲隐藏的 Widget 写在此 */
  mTextView01.setVisibility(View.INVISIBLE);
  mEditText01.setVisibility(View.INVISIBLE);
}

/* 开始 ScreenSaver */
private void showScreenSaver()
{
  /* 屏幕保护之后要做的事件写在此*/
  if(intDrawable>4)
  {
    intDrawable = 0;
  }

  DisplayMetrics dm=new DisplayMetrics();
  getWindowManager().getDefaultDisplay().getMetrics(dm);
  screenWidth = dm.widthPixels;
  screenHeight = dm.heightPixels;
  Bitmap bmp=BitmapFactory.decodeResource(getResources(),
  screenDrawable[intDrawable]);
```

(12) 通过 Matrix 设置比例，使用 Matrix.postScale 设置维度 ReSize，通过 resizedBitmap 对象设置图文件至屏幕分辨率，新建 Drawable 对象 myNewBitmapDrawable 用于放大图文件至全屏幕，通过 setVisibility(View.VISIBLE)使 ImageView 可见。具体代码如下所示：

```
  /* Matrix 比例 */
  float scaleWidth = ((float) screenWidth) / bmp.getWidth();
  float scaleHeight = ((float) screenHeight) / bmp.getHeight() ;

  Matrix matrix = new Matrix();
```

```
  /* 使用 Matrix.postScale 设置维度 ReSize */
  matrix.postScale(scaleWidth, scaleHeight);
  /* ReSize 图文件至屏幕分辨率 */
  Bitmap resizedBitmap = Bitmap.createBitmap
  (
    bmp,0,0,bmp.getWidth(),bmp.getHeight(),matrix,true
  );

  /* 新建 Drawable 放大图文件至全屏幕 */
  BitmapDrawable myNewBitmapDrawable =
      new BitmapDrawable(resizedBitmap);
  mImageView01.setImageDrawable(myNewBitmapDrawable);

  /* 使 ImageView 可见 */
  mImageView01.setVisibility(View.VISIBLE);

  /* 每间隔设置秒数置换图片 ID，在下一个 runnable2 才会生效 */
  if(intCounter2%intSecondsToChange==0)
  {
    intDrawable++;
  }
}
```

(13) 定义方法 onUserWakeUpEvent()，实现解锁和加密处理。具体代码如下所示：

```
public void onUserWakeUpEvent()
{
  if(bIfRunScreenSaver==true)
  {
    try
    {
      /* LayoutInflater.from 取得此 Activity 的 context */
      mInflater01 = LayoutInflater.from(example3.this);

      /* 创建解锁密码使用 View 的 Layout */
      mView01 = mInflater01.inflate(R.layout.securescreen, null);

      /* 在对话框中唯一的 EditText 等待输入解锁密码 */
      mEditText02 =
      (EditText) mView01.findViewById(R.id.myEditText2);

      /* 创建 AlertDialog */
      new AlertDialog.Builder(this)
      .setView(mView01)
      .setPositiveButton("OK",
      new DialogInterface.OnClickListener()
      {
        public void onClick(DialogInterface dialog, int whichButton)
        {
          /* 比较输入的密码与原 Activity 里的设置是否相符 */
          if(mEditText01.getText().toString().equals
            (mEditText02.getText().toString()))
          {
```

```
            /* 当密码正确才真得解锁屏幕保护装置 */
            resetScreenSaverListener();
          }
        }
      }).show();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

(14) 定义 updateUserActionTime()，用于统计用户单击键盘或屏幕的时间间隔。先取得单击按键事件时的系统 Time Millis，然后重新计算单击按键距离上一次静止的时间间距。具体代码如下所示：

```
public void updateUserActionTime()
{
  /* 取得单击按键事件时的系统 Time Millis */
  Date timeNow = new Date(System.currentTimeMillis());

  /* 重新计算单击按键距离上一次静止的时间间距 */
  timePeriod =
  (long)timeNow.getTime() - (long)lastUpdateTime.getTime();
  lastUpdateTime.setTime(timeNow.getTime());
}
```

(15) 定义方法 resetScreenSaverListener()，用于重新设置屏幕，实现的具体流程如下。

第 1 步：删除现有的 Runnable，然后取得单击按键事件时的系统 Time Millis。

第 2 步：重新计算单击按键距离上一次静止的时间间距。

第 3 步：通过 bIfRunScreenSaver 取消屏保并恢复原来的 Layout Visible。

具体代码如下所示：

```
public void resetScreenSaverListener()
{
  /* 删除现有的 Runnable */
  mHandler01.removeCallbacks(mTasks01);
  mHandler02.removeCallbacks(mTasks02);

  /* 取得单击按键事件时的系统 Time Millis */
  Date timeNow = new Date(System.currentTimeMillis());
  /* 重新计算单击按键距离上一次静止的时间间距 */
  timePeriod =
  (long)timeNow.getTime() - (long)lastUpdateTime.getTime();
  lastUpdateTime.setTime(timeNow.getTime());

  /* for Runnable2，取消屏幕保护 */
  bIfRunScreenSaver = false;

  /* 重置 Runnable1 与 Runnable1 的 Counter */
  intCounter1 = 0;
```

```
    intCounter2 = 0;

    /* 恢复原来的 Layout Visible*/
    recoverOriginalLayout();

    /* 重新 postDelayed()新的 Runnable */
    mHandler01.postDelayed(mTasks01, intervalKeypadeSaver);
  }
```

(16) 定义 onKeyDown(int keyCode, KeyEvent event)，用于监听用户的触摸单击事件。具体代码如下所示：

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event)
{
  // TODO Auto-generated method stub
  if(bIfRunScreenSaver==true && keyCode!=4)
  {
    /* 当屏幕保护程序正在运行中，触动解除屏幕保护程序 */
    onUserWakeUpEvent();
  }
  else
  {
    /* 更新 User 未触动手机的时间戳记 */
    updateUserActionTime();
  }
  return super.onKeyDown(keyCode, event);
}
@Override
public boolean onTouchEvent(MotionEvent event)
{
  // TODO Auto-generated method stub
  if(bIfRunScreenSaver==true)
  {
    /* 当屏幕保护程序正在运行中，触动解除屏幕保护程序 */
    onUserWakeUpEvent();
  }
  else
  {
    /* 更新 User 未触动手机的时间戳记 */
    updateUserActionTime();
  }
  return super.onTouchEvent(event);
}

@Override
protected void onResume()
{
  // TODO Auto-generated method stub
  mHandler01.postDelayed(mTasks01, intervalKeypadeSaver);
  super.onResume();
}
```

(17) 定义方法 onPause()，用于删除运行中的运行线程 mHandler01、mHandler02、mHandler03 和 mHandler04。具体代码如下所示：

```
@Override
protected void onPause()
{
  // TODO Auto-generated method stub

  try
  {
    /* 删除运行中的运行线程 */
    mHandler01.removeCallbacks(mTasks01);
    mHandler02.removeCallbacks(mTasks02);
    mHandler03.removeCallbacks(mTasks03);
    mHandler04.removeCallbacks(mTasks04);
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
  super.onPause();
 }
}
```

至此整个展示结束。执行后如果超过 5 秒，不动键盘或屏幕则会进入屏保状态，执行效果如图 12-2 所示。另外也可以设置屏保密码，当输入正确的密码后才能解除屏保，密码解锁如图 12-3 所示。



图 12-2　执行效果

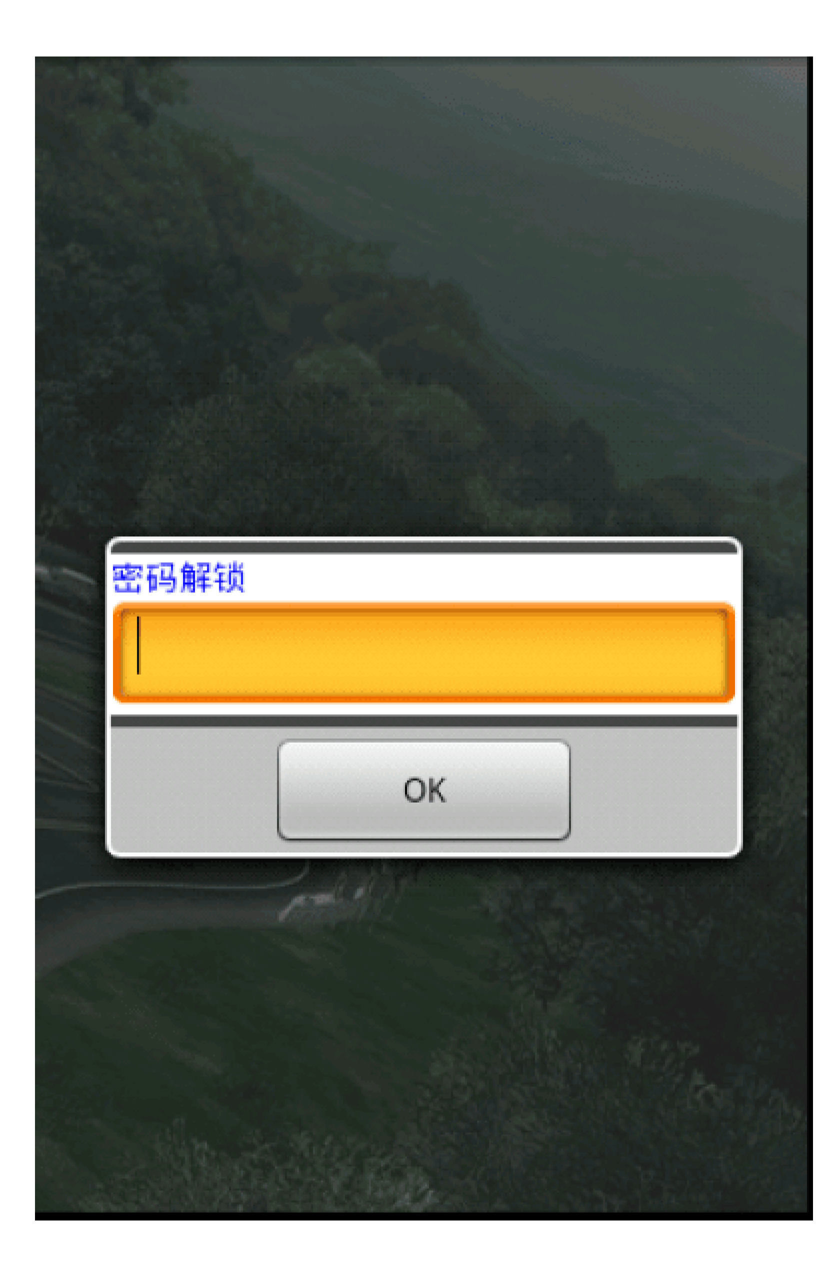


图 12-3　密码解锁

12.4 触摸移动图片

题　目	目　的	源码路径
题目 3	在 Android 中实现触摸移动照片	"光盘:\daima\12\example4"文件夹

12.4.1 我的想法

在触摸屏手机中，点击移动照片的功能十分常见。细想之下，其实实现起来非常简单。可以设计使用 Drawable 照片的 ImageView，暂时将照片在程序运行的开始时放在屏幕中央。通过 onTouchEvent 来处理点击、拖动、放开等事件来完成拖动图片的功能。通过设置了 ImageView 的 onClick-Listener 让用户在点击图片的同时，恢复图片到初始位置。

12.4.2 具体实现

编写的主程序文件是 example4.java，其具体实现代码如下。

(1) 先通过 DisplayMetrics 取得屏幕对象，然后通过 intScreenX 和 intScreenY 取得屏幕解析像素，最后分别设置图片的宽、高。具体代码如下所示：

```
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
/* 取得屏幕对象 */
DisplayMetrics dm = new DisplayMetrics();
getWindowManager().getDefaultDisplay().getMetrics(dm);

/* 取得屏幕解析像素 */
intScreenX = dm.widthPixels;
intScreenY = dm.heightPixels;

/* 设置图片的宽高 */
intWidth = 100;
intHeight = 100;
```

(2) 通过 findViewById 构造器创建 ImageView 对象，然后将图片从 Drawable 赋值给 ImageView 来呈现，并通过 RestoreButton()初始化按钮位置居中。具体代码如下所示：

```
/*通过 findViewById 构造器创建 ImageView 对象*/
mImageView01 =(ImageView) findViewById(R.id.myImageView1);
/*将图片从 Drawable 赋值给 ImageView 来呈现*/
mImageView01.setImageResource(R.drawable.baby);
/* 初始化按钮位置居中 */
RestoreButton();
```

(3) 定义 setOnClickListener(new Button.OnClickListener()，用于当单击 ImageView 时还原初始位置。具体代码如下所示：

```
/* 当单击 ImageView，还原初始位置 */
```

```
  mImageView01.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View v)
    {
      RestoreButton();
    }
  });
```

(4) 定义 onTouchEvent(MotionEvent event)覆盖触控事件。首先取得手指触控屏幕的位置，然后实现触控事件的处理，即实现点击屏幕、移动位置、离开屏幕的处理。具体代码如下所示：

```
/*覆盖触控事件*/
@Override
public boolean onTouchEvent(MotionEvent event)
{
  /*取得手指触控屏幕的位置*/
  float x = event.getX();
  float y = event.getY();

  try
  {
    /*触控事件的处理*/
    switch (event.getAction())
    {
      /*单击屏幕*/
      case MotionEvent.ACTION_DOWN:
        picMove(x, y);
          break;
      /*移动位置*/
      case MotionEvent.ACTION_MOVE:
        picMove(x, y);
          break;
      /*离开屏幕*/
      case MotionEvent.ACTION_UP:
        picMove(x, y);
          break;
    }
  }catch(Exception e)
    {
      e.printStackTrace();
    }
  return true;
}
```

(5) 定义 picMove(float x, float y)，用于实现移动图片。具体代码如下所示：

```
/*移动图片的方法*/
private void picMove(float x, float y)
{
  /*默认微调图片与指针的相对位置*/
  mX=x-(intWidth/2);
  mY=y-(intHeight/2);
```

```
  /*防止图片超过屏幕的相关处理*/
  /*防止屏幕向右超过屏幕*/
  if((mX+intWidth)>intScreenX)
  {
    mX = intScreenX-intWidth;
  }
  /*防止屏幕向左超过屏幕*/
  else if(mX<0)
  {
    mX = 0;
  }
  /*防止屏幕向下超过屏幕*/
  else if ((mY+intHeight)>intScreenY)
  {
    mY=intScreenY-intHeight;
  }
  /*防止屏幕向上超过屏幕*/
  else if (mY<0)
  {
    mY = 0;
  }
  /*通过 log 来查看图片位置*/
  Log.i("jay", Float.toString(mX)+","+Float.toString(mY));
  /* 以 setLayoutParams 方法，重新安排 Layout 上的位置 */
  mImageView01.setLayoutParams
  (
    new AbsoluteLayout.LayoutParams
    (intWidth,intHeight,(int) mX,(int)mY)
  );
}
```

(6) 定义 RestoreButton()，用于还原 ImageView 位置的事件处理。具体代码如下所示：

```
/* 还原 ImageView 位置的事件处理 */
public void RestoreButton()
{
  intDefaultX = ((intScreenX-intWidth)/2);
  intDefaultY = ((intScreenY-intHeight)/2);
  /*Toast 还原位置坐标*/
  mMakeTextToast
  (
    "("+
    Integer.toString(intDefaultX)+
    ","+
    Integer.toString(intDefaultY)+")",true
  );

  /* 以 setLayoutParams 方法，重新安排 Layout 上的位置 */
  mImageView01.setLayoutParams
  (
    new AbsoluteLayout.LayoutParams
    (intWidth,intHeight,intDefaultX,intDefaultY)
  );
}
```

(7)　定义 mMakeTextToast(String str, boolean isLong)，用于自定义一发出信息的方法。具体代码如下所示：

```
/*自定义一发出信息的方法*/
public void mMakeTextToast(String str, boolean isLong)
{
  if(isLong==true)
  {
    Toast.makeText(example4.this, str, Toast.LENGTH_LONG).show();
  }
  else
  {
    Toast.makeText(example4.this, str, Toast.LENGTH_SHORT).show();
  }
}
}
```

至此整个展示结束，执行后的效果如图 12-4 所示，能在屏幕中通过鼠标单击来移动指定图片的位置，如图 12-5 所示。



图 12-4　执行效果

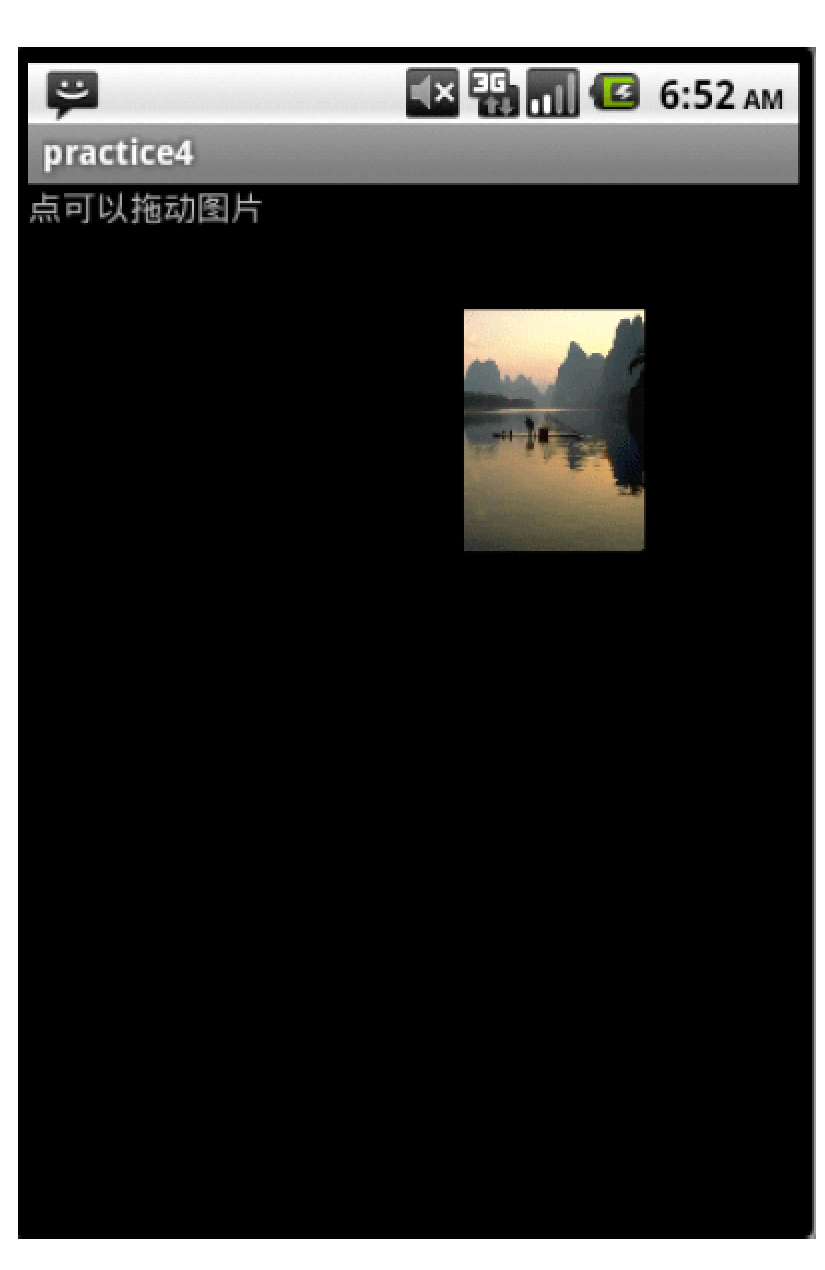


图 12-5　移动图片的位置

12.5　显示存储卡中的照片

题　目	目　的	源码路径
题目 4	在 Android 中获取存储卡中的图片	“光盘:\daima\12\example5”文件夹

12.5.1 我的想法

我决定定义一个获取文件列表的函数 getSD()，这样可以将内存中扫描过的照片以 File List<String>的方式存储，再利用定义的 ImageAdapter 来初始化 Gallery 对象，最后将存储卡中的图片加载到 Gallery Widget 中。

12.5.2 具体实现

编写的主程序文件是 example5.java，其具体实现代码如下。

(1) 定义 List<String> getSD()，通过 ArrayList 作为自定义 SD 卡访问图片文件列表时使用，并将所有档案存放到 ArrayList 中。具体代码如下所示：

```
private List<String> getSD()
{
  /* 设定目前所在路径 */
  List<String> it=new ArrayList<String>();
  File f=new File("/sdcard/");
  File[] files=f.listFiles();

  /* 将所有档案存放到 ArrayList 中 */
  for(int i=0;i<files.length;i++)
  {
    File file=files[i];
    if(getImageFile(file.getPath()))
      it.add(file.getPath());
  }
  return it;
}
```

(2) 定义 getImageFile(String fName)获取存储卡内的图片文件，并根据扩展类型决定 MimeType。具体代码如下所示：

```
private boolean getImageFile(String fName)
{
  boolean re;

  /* 取得扩展名 */
  String end=fName.substring(fName.lastIndexOf(".")+1,
              fName.length()).toLowerCase();

  /* 根据扩展类型决定 MimeType */
  if(end.equals("jpg")||end.equals("gif")||end.equals("png")
        ||end.equals("jpeg")||end.equals("bmp"))
  {
    re=true;
  }
  else
  {
    re=false;
```

```
    }
    return re;
  }
```

(3) 改写 BaseAdapter 自定义 ImageAdapter class，先声明变量和 ImageAdapter 构造器，然后获取 Gallery 信息。具体代码如下所示：

```
/*改写 BaseAdapter 自定义 ImageAdapter class*/
public class ImageAdapter extends BaseAdapter
{
  /*声明变量*/
  int mGalleryItemBackground;
  private Context mContext;
  private List<String> lis;

  /*ImageAdapter 构造器*/
  public ImageAdapter(Context c,List<String> li)
  {
    mContext = c;
    lis=li;
    /* 使用?res/values/attrs.xml 中的<declare-styleable>定义
    * 的 Gallery 属性。*/
    TypedArray a = obtainStyledAttributes(R.styleable.Gallery);
    /*取得 Gallery 属性的 Index id*/
    mGalleryItemBackground = a.getResourceId(
        R.styleable.Gallery_android_galleryItemBackground, 0);
    /*让对象的 styleable 属性能够反复使用*/
    a.recycle();
  }
```

(4) 定义三个需要覆盖的方法，其中 getCount 用于返回图片数目；getItem 用于返回位置；getView 用于返回 View 对象。具体代码如下所示：

```
  /*定义要覆盖的方法 getCount，返回图片数目*/
  public int getCount()
  {
    return lis.size();
  }

  /*定义要覆盖的方法 getItem，用于返回位置*/
  public Object getItem(int position)
  {
    return position;
  }

  /*定义要覆盖的方法 getItemId，用于返回 position */
  public long getItemId(int position)
  {
    return position;
  }

  /*定义要覆盖的方法 getView，用于返回 View 对象*/
  public View getView(int position, View convertView, ViewGroup parent)
```

```
    {
      /*产生 ImageView 对象*/
      ImageView i = new ImageView(mContext);
      /*设定图片给 imageView 对象*/
      Bitmap bm = BitmapFactory.decodeFile(lis.
                        get(position).toString());
      i.setImageBitmap(bm);
      /*重新设定图片的宽高*/
      i.setScaleType(ImageView.ScaleType.FIT_XY);
      /*重新设定 Layout 的宽高*/
      i.setLayoutParams(new Gallery.LayoutParams(136, 88));
      /*设定 Gallery 背景图*/
      i.setBackgroundResource(mGalleryItemBackground);
      /*传送 imageView 物件*/
      return i;
    }
  }
}
```

至此整个展示结束，执行后将会获取显示存储卡内的图片信息，并以 Gallery 的样式显示出来，执行效果如图 12-6 所示。鼠标单击后能够滑动显示图片，如图 12-7 所示。



图 12-6　执行效果



图 12-7　滑动显示图片

接下来详细讲解本实例的测试过程。

(1)　创建虚拟 SD 卡

用 cmd 进入到 android SDK 的 Tools 目录下执行 mksdcard 创建。例如，我的 tools 目录为：

```
E:\skyland\android-sdk-windows\tools>
```

创建的命令为如下代码：

```
E:\skyland\android-sdk-windows\tools>mksdcard 128M sdcard.img
```

其中，第一个参数为要创建的 sdcard 容量的大小(容量大小自己决定)，第二个参数为 sdcard 的名字。

(2)　启动带 sdcard 的 Android 模拟器

其实也可以不用 cmd 命令，直接在 Eclipse 中启用，具体方法如下。

第 1 步：右键单击实例工程，在弹出的菜单中依次选择 Run As | Run Configurations 菜单命令，如图 12-8 所示。

第 2 步：在弹出的对话框中切换到 Target 选项卡，在最后一行中输入如下字符：

```
-sdcard c:\sdcard.img
```

如图 12-9 所示。

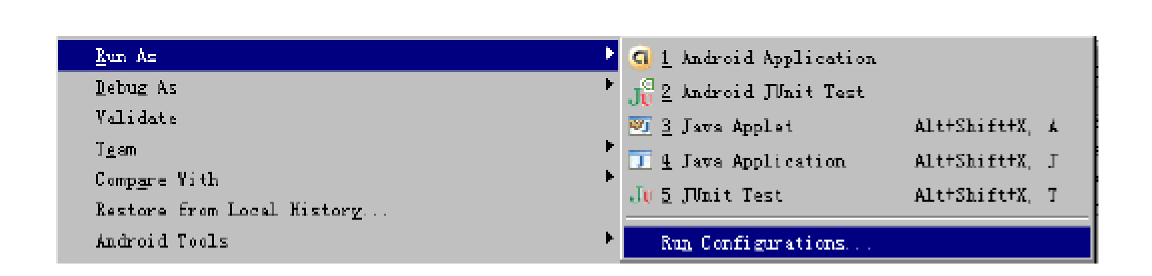


图 12-8　菜单命令

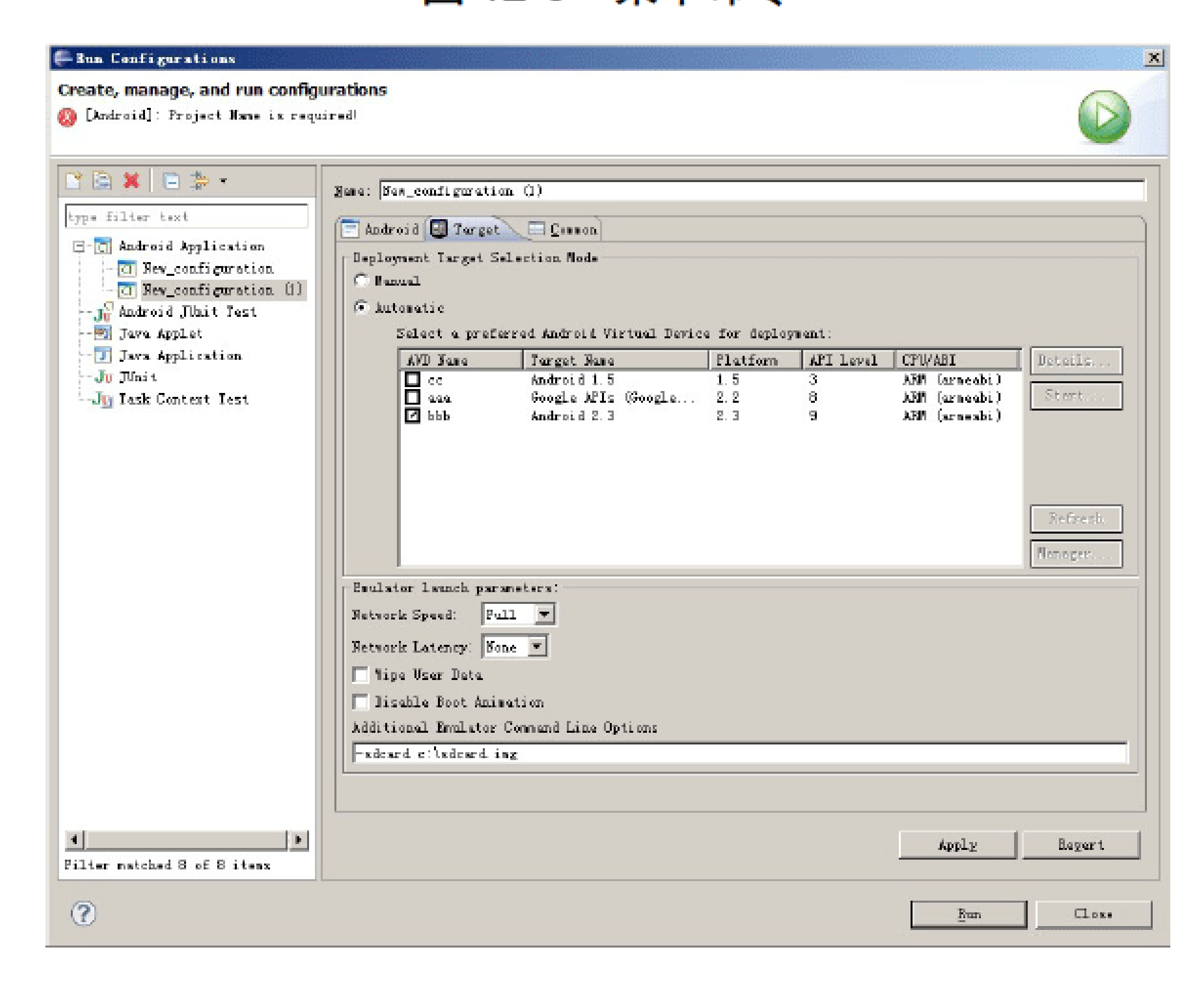


图 12-9　输入-sdcard c:\sdcard.img

(3)　添加文件到创建的 SD 卡

经过以上两步操作后，模拟器已经运行了，下面开始添加文件到创建的 SD 卡，有以下两种操作方法。

第一种：采用 Eclipse 菜单实现。

在 Eclipse 中打开 DDMS 视图，切换到 File Explor 选项卡，单击目录下的 sdcard 文件，如图 12-10 所示。

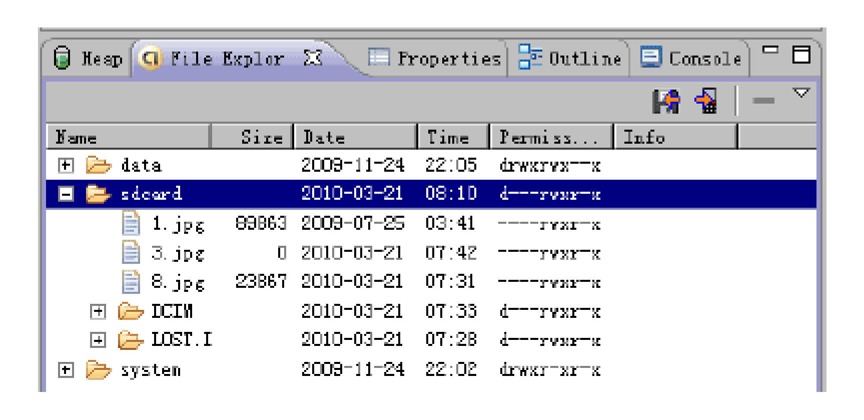


图 12-10　File Explor 选项卡

通过单击图 12-10 中的按钮，可以上传本地文件到虚拟存储卡中。

第二种：用 cmd 命令。

通过 cmd 命令到 android SDK 的 Tools 目录下，用 adb push 命令进行添加，代码格式如下：

```
E:\skyland\android-sdk-windows\tools>adb push new.JPG /sdcard
```

其中，第一个参数为要加入的图片全名，如果名字中间有空格，要用双引号将其括起来。例如下面的代码：

```
adb push "c:\1.jpg" /sdcard
```

第二个参数就是刚创建的 sdcard 了。

12.6 调节音量大小

题　目	目　的	源码路径
题目 5	在 Android 中调节音量的大小	“光盘:\daima\12\example7”文件夹

12.6.1 我的想法

在 Android 中可以调节手机的音量大小，看似复杂其实实现起来很简单。可以用 Android API 中 AudioManager 提供的方法来实现，即可以在程序中控制手机音量的大小，也可以切换声音的模式为震动或静音。我将本例使用的素材图片存放在“res\drawable”目录下。

12.6.2 具体实现

编写的主程序文件是 example7.java，其具体实现代码如下。

(1) 分别设置初始的手机音量和声音模式。具体代码如下所示：

```
/* 设置初始的手机音量 */
volume=audioMa.getStreamVolume(AudioManager.STREAM_RING);
myProgress.setProgress(volume);
/* 设置初始的声音模式 */
int mode=audioMa.getRingerMode();
if(mode==AudioManager.RINGER_MODE_NORMAL)
{
  myImage.setImageDrawable(getResources()
                           .getDrawable(R.drawable.normal));
}
else if(mode==AudioManager.RINGER_MODE_SILENT)
{
  myImage.setImageDrawable(getResources()
                           .getDrawable(R.drawable.mute));
}
else if(mode==AudioManager.RINGER MODE VIBRATE)
{
  myImage.setImageDrawable(getResources()
                           .getDrawable(R.drawable.vibrate));
}
```

(2) 设置音量调小按钮 downButton 的处理事件 setOnClickListener，先设置音量调小声一格，然后设置调整后的声音模式。具体代码如下所示：

```
/* 音量调小声的 Button */
downButton.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View arg0)
  {
    /* 设置音量调小声一格 */
```

```
        audioMa.adjustVolume(AudioManager.ADJUST_LOWER, 0);
        volume=audioMa.getStreamVolume(AudioManager.STREAM_RING);
        myProgress.setProgress(volume);
        /* 设置调整后声音模式 */
        int mode=audioMa.getRingerMode();
        if(mode==AudioManager.RINGER_MODE_NORMAL)
        {
          myImage.setImageDrawable(getResources()
                           .getDrawable(R.drawable.normal));
        }
        else if(mode==AudioManager.RINGER_MODE_SILENT)
        {
          myImage.setImageDrawable(getResources()
                           .getDrawable(R.drawable.mute));
        }
        else if(mode==AudioManager.RINGER_MODE_VIBRATE)
        {
          myImage.setImageDrawable(getResources()
                          .getDrawable(R.drawable.vibrate));
        }
      }
    });
```

(3) 设置音量调大按钮 upButton 的处理事件 setOnClickListener，先设置音量调大声一格，然后设置调整后的声音模式。具体代码如下所示：

```
    /* 音量调大声的 Button */
    upButton.setOnClickListener(new Button.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
        /* 设置音量调大声一格 */
        audioMa.adjustVolume(AudioManager.ADJUST_RAISE, 0);
        volume=audioMa.getStreamVolume(AudioManager.STREAM_RING);
        myProgress.setProgress(volume);
        /* 设置调整后的声音模式 */
        int mode=audioMa.getRingerMode();
        if(mode==AudioManager.RINGER_MODE_NORMAL)
        {
          myImage.setImageDrawable(getResources()
                           .getDrawable(R.drawable.normal));
        }
        else if(mode==AudioManager.RINGER_MODE_SILENT)
        {
          myImage.setImageDrawable(getResources()
                           .getDrawable(R.drawable.mute));
        }
        else if(mode==AudioManager.RINGER_MODE_VIBRATE)
        {
          myImage.setImageDrawable(getResources()
                          .getDrawable(R.drawable.vibrate));
```

```
        }
      }
    });
```

(4) 设置调整正常铃声模式按钮 normalButton 的处理事件 setOnClickListener，先设置铃声模式为 NORMAL，然后设置音量与声音模式。具体代码如下所示：

```
    /* 调整铃声模式为正常模式的 Button */
    normalButton.setOnClickListener(new Button.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
        /* 设置铃声模式为 NORMAL */
        audioMa.setRingerMode(AudioManager.RINGER_MODE_NORMAL);
        /* 设置音量与声音模式 */
        volume=audioMa.getStreamVolume(AudioManager.STREAM_RING);
        myProgress.setProgress(volume);
        myImage.setImageDrawable(getResources()
                                .getDrawable(R.drawable.normal));
      }
    });
```

(5) 设置调整静音铃声模式按钮 muteButton 的处理事件 setOnClickListener，先设置铃声模式为 SILENT，然后设置音量与声音状态。具体代码如下所示：

```
    /* 调整铃声模式为静音模式的 Button */
    muteButton.setOnClickListener(new Button.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
        /* 设置铃声模式为 SILENT */
        audioMa.setRingerMode(AudioManager.RINGER_MODE_SILENT);
        /* 设置音量与声音状态 */
        volume=audioMa.getStreamVolume(AudioManager.STREAM RING);
        myProgress.setProgress(volume);
        myImage.setImageDrawable(getResources()
                                .getDrawable(R.drawable.mute));
      }
    });
```

(6) 设置调整震动铃声模式按钮 vibrateButton 的处理事件 setOnClickListener，先设置铃声模式为 VIBRATE，然后设置音量与声音状态。具体代码如下所示：

```
    /* 调整铃声模式为震动模式的 Button */
    vibrateButton.setOnClickListener(new Button.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
        /* 设置铃声模式为 VIBRATE */
        audioMa.setRingerMode(AudioManager.RINGER_MODE_VIBRATE);
```

```
        /* 设置音量与声音状态 */
        volume=audioMa.getStreamVolume(AudioManager.STREAM_RING);
        myProgress.setProgress(volume);
        myImage.setImageDrawable(getResources()
                            .getDrawable(R.drawable.vibrate));
      }
    });
  }
}
```

至此整个展示结束，执行后将会显示一个音量调节界面，执行效果如图 12-11 所示。



图 12-11　执行效果

12.7　播放 MP3 文件

题　目	目　的	源码路径
题目 6	在 Android 手机中播放一个 MP3 文件	“光盘:\daima\12\example8”文件夹

12.7.1　我的想法

移动手机播放 MP3 的功能十分常见，在屏幕中插入 3 个按钮，分别用于播放、暂停和停止。当单击播放按钮后会从指定的手机资源中获取 mp3 文件，并执行播放处理。在具体实现上，先添加一个 MediaPlayer 对象，使用 MediaPlayer.creat()方法来创建播放器资源，然后通过 MediaPlay.creat()、MediaPlay.stop()和 MediaPlay.pause()方法分别来实现开始、停止和暂停功能。为了处理按钮所需要的各个事件，需要覆盖各个 ImageButton 的 onClick()，用于通过按钮来控制 MediaPlayer 的状态。

12.7.2　具体实现

编写的主程序文件是 example8.java，其具体实现代码如下。

(1)　引入主布局文件 main.xml，通过 findViewById 构造器创建 TextView 与 ImageView 对象，然后创建 MediaPlayer 对象，并将音乐以 Import 的方式存储在 res/raw/always.mp3 中，最后创建 MediaPlayer 对象。具体代码如下所示：

```
  /** Called when the activity is first created. */
```

```
@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  /*通过 findViewById 构造器创建 TextView 与 ImageView 对象*/
  mButton01 =(ImageButton) findViewById(R.id.myButton1);
  mButton02 =(ImageButton) findViewById(R.id.myButton2);
  mButton03 =(ImageButton) findViewById(R.id.myButton3);
  mTextView01 = (TextView) findViewById(R.id.myTextView1);

  /* onCreate 时创建 MediaPlayer 对象 */
  mMediaPlayer01 = new MediaPlayer();
  /* 将音乐以 Import 的方式存储在 res/raw/always.mp3 */
  mMediaPlayer01 = MediaPlayer.create(example8.this, R.raw.big);
```

(2) 运行播放音乐的按钮。首先覆盖 OnClick 事件；然后在 MediaPlayer 取得播放资源与 stop()之后开始或回复播放；最后改变 TextView 为开始播放状态。具体代码如下所示：

```
  /* 运行播放音乐的按钮 */
  mButton01.setOnClickListener(new ImageButton.OnClickListener()
  {
    @Override
    /*覆盖 OnClick 事件*/
    public void onClick(View v)
    {
      // TODO Auto-generated method stub
      try
      {
        if (mMediaPlayer01 != null)
        {
          mMediaPlayer01.stop();
        }
        /*在 MediaPlayer 取得播放资源与 stop()之后
         * 要准备 Playback 的状态前一定要使用 MediaPlayer.prepare()*/
        mMediaPlayer01.prepare();
        /*开始或回复播放*/
        mMediaPlayer01.start();
        /*改变 TextView 为开始播放状态*/
        mTextView01.setText(R.string.str_start);
      }
      catch (Exception e)
      {
        // TODO Auto-generated catch block
        mTextView01.setText(e.toString());
        e.printStackTrace();
      }
    }
  });
```

(3) 定义停止播放处理事件，通过 mMediaPlayer01.stop()停止播放 MP3，改变 TextView 为

停止播放状态。具体代码如下所示：

```
/* 停止播放 */
mButton02.setOnClickListener(new ImageButton.OnClickListener()
{
  @Override
  public void onClick(View arg0)
  {
    // TODO Auto-generated method stub
    try
    {
      if (mMediaPlayer01 != null)
      {
        /*停止播放*/
        mMediaPlayer01.stop();
        /*改变 TextView 为停止播放状态*/
        mTextView01.setText(R.string.str_close);
      }

    }
    catch (Exception e)
    {
      // TODO Auto-generated catch block
      mTextView01.setText(e.toString());
      e.printStackTrace();
    }
  }
});
```

(4)　定义暂停播放处理事件，首先判断是否处于暂停状态，否则暂停。具体代码如下所示：

```
/* 暂停播放 */
mButton03.setOnClickListener(new ImageButton.OnClickListener()
{
  @Override
  public void onClick(View arg0)
  {
    // TODO Auto-generated method stub
    try
    {
      if (mMediaPlayer01 != null)
      {
        /*是否为暂停状态=否*/
        if(bIsPaused==false)
        {
          /*暂停播放*/
          mMediaPlayer01.pause();
          /*设置 Flag 为 true 表示 Player 状态为暂停*/
          bIsPaused = true;
          /*改变 TextView 为暂停播放*/
          mTextView01.setText(R.string.str_pause);
        }
```

```
          /*是否为暂停状态=是*/
          else if(bIsPaused==true)
          {
            /*回复播出状态*/
            mMediaPlayer01.start();
            /*设置 Flag 为 false 表示 Player 状态为非暂停状态*/
            bIsPaused = false;
            /*改变 TextView 为开始播放*/
            mTextView01.setText(R.string.str_start);
          }
        }
      }
      catch (Exception e)
      {
        // TODO Auto-generated catch block
        mTextView01.setText(e.toString());
        e.printStackTrace();
      }
    }
  });
```

(5) 通过 Media Player OnCompletionLister 事件来监听是否播放完毕，如果播放完毕，则改变 TexView 的显示信息为“播放完毕”。具体代码如下所示：

```
  /* 当 MediaPlayer.OnCompletionLister 会运行的 Listener */
  mMediaPlayer01.setOnCompletionListener(
    new MediaPlayer.OnCompletionListener()
  {
    // @Override
    /*覆盖文件播出完毕事件*/
    public void onCompletion(MediaPlayer arg0)
    {
      try
      {
        /*解除资源与 MediaPlayer 的赋值关系
         * 让资源可以为其他程序利用*/
        mMediaPlayer01.release();
        /*改变 TextView 为播放结束*/
        mTextView01.setText(R.string.str_OnCompletionListener);
      }
      catch (Exception e)
      {
        mTextView01.setText(e.toString());
        e.printStackTrace();
      }
    }
  });
```

(6) 当 MediaPlayer.OnErrorListener 运行 Listener 时，覆盖错误处理事件，当发生错误时也解除资源与 MediaPlayer 的赋值。具体代码如下所示：

```
    /* 当 MediaPlayer.OnErrorListener 会运行的 Listener */
    mMediaPlayer01.setOnErrorListener(new MediaPlayer.OnErrorListener()
    {
      @Override
      /*覆盖错误处理事件*/
      public boolean onError(MediaPlayer arg0, int arg1, int arg2)
      {
        // TODO Auto-generated method stub
        try
        {
          /*发生错误时也解除资源与 MediaPlayer 的赋值*/
          mMediaPlayer01.release();
          mTextView01.setText(R.string.str_OnErrorListener);
        }
        catch (Exception e)
        {
          mTextView01.setText(e.toString());
          e.printStackTrace();
        }
        return false;
      }
    });
  }
```

(7) 覆盖主程序暂停状态事件，在主程序暂停时解除资源与 MediaPlayer 的赋值关系，通过 Catch 实现异常处理。具体代码如下所示：

```
  @Override
  /*覆盖主程序暂停状态事件*/
  protected void onPause()
  {
    // TODO Auto-generated method stub
    try
    {
      /*在主程序暂停时解除资源与 MediaPlayer 的赋值关系*/
      mMediaPlayer01.release();
    }
    catch (Exception e)
    {
      mTextView01.setText(e.toString());
      e.printStackTrace();
    }
    super.onPause();
  }
}
```

至此整个展示结束，执行后会在屏幕中通过三个播放按钮播放指定的 MP3 文件，执行效果如图 12-12 所示。

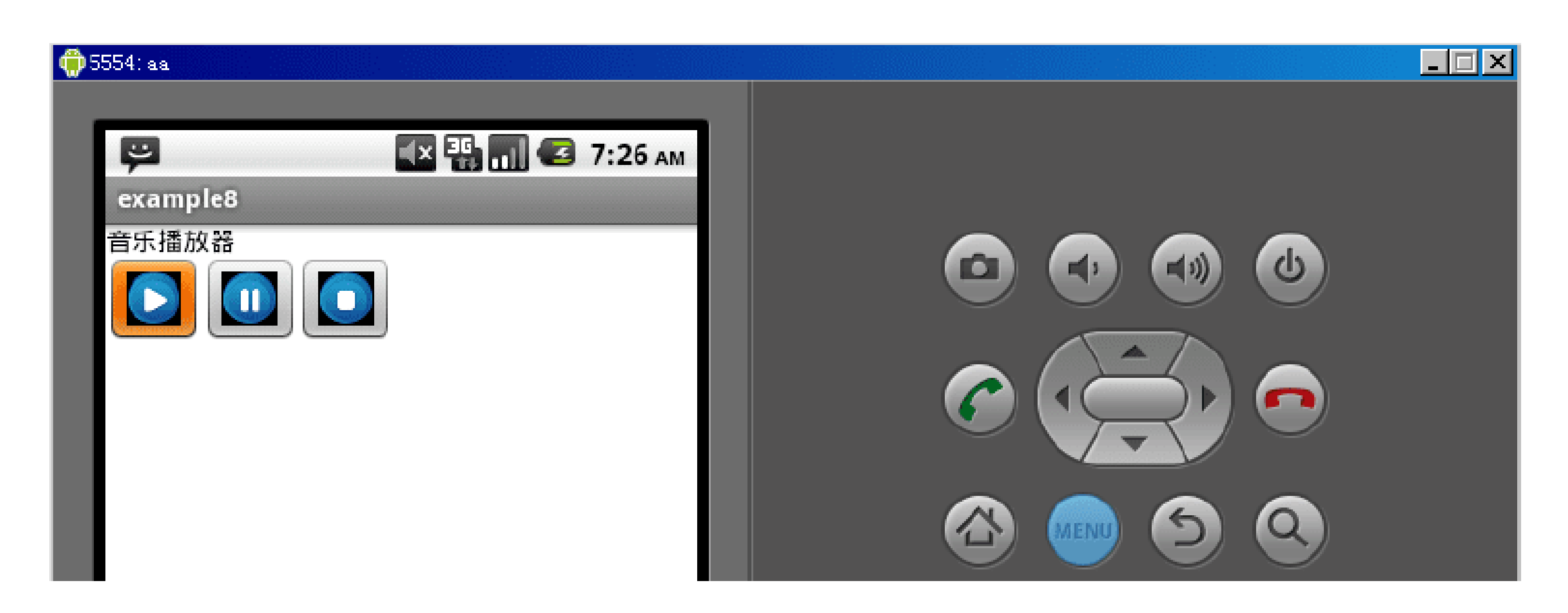


图 12-12　执行效果

12.8　录 音 处 理

题　目	目　的	源码路径
题目 7	在 Android 手机中实现录音处理	“光盘:\daima\12\example9”文件夹

12.8.1　我的想法

录音处理也是移动手机的重要功能之一，我决定插入四个按钮分别用于录音、停止录音、播放录音和删除录音。为了能够不限制录音的长度，现将录音暂时保存到存储卡，当录音完毕后再将录音文件显示在 ListView，单击文件后，可以播放或删除录音文件。

12.8.2　具体实现

编写的主程序文件是 example9.java，其具体实现代码如下。

(1)　通过 sdCardExit 判断 SD Card 是否插入，然后获取 SD Card 路径作为录音的文件位置，并取得 SD Card 目录中的所有.amr 文件，最后将 ArrayAdapter 添加到 ListView 对象中。具体代码如下所示：

```
/* 判断 SD Card 是否插入 */
sdCardExit = Environment.getExternalStorageState().equals(
    android.os.Environment.MEDIA_MOUNTED);
/* 取得 SD Card 路径作为录音的文件位置 */
if (sdCardExit)
  myRecAudioDir = Environment.getExternalStorageDirectory();
/* 取得 SD Card 目录里的所有.amr 文件 */
getRecordFiles();
adapter = new ArrayAdapter<String>(this,
    R.layout.my_simple_list_item, recordFiles);
/* 将 ArrayAdapter 添加到 ListView 对象中 */
myListView1.setAdapter(adapter);
```

(2)　设置单击录音按钮后的录音处理事件。先创建录音文件，然后设置录音来源为麦克风，最后通过 myTextView1.setText(“录音中”)设置录音过程显示的提示文本。具体代码如下所示：

```
/* 录音 */
```

```
myButton1.setOnClickListener(new ImageButton.OnClickListener()
{
  @Override
  public void onClick(View arg0)
  {
    try
    {
      if (!sdCardExit)
      {
        Toast.makeText(example9.this, "请插入 SD Card",
            Toast.LENGTH_LONG).show();
        return;
      }
      /* 创建音频文件 */
      myRecAudioFile = File.createTempFile(strTempFile, ".amr",
          myRecAudioDir);
      mMediaRecorder01 = new MediaRecorder();
      /* 设置录音来源为麦克风 */
      mMediaRecorder01
          .setAudioSource(MediaRecorder.AudioSource.MIC);
      mMediaRecorder01
          .setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
      mMediaRecorder01
          .setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
      mMediaRecorder01.setOutputFile(myRecAudioFile
          .getAbsolutePath());
      mMediaRecorder01.prepare();
      mMediaRecorder01.start();
      myTextView1.setText("录音中");
      myButton2.setEnabled(true);
      myButton3.setEnabled(false);
      myButton4.setEnabled(false);
      isStopRecord = false;
    }
    catch (IOException e)
    {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
});
```

(3)　设置单击停止按钮后的处理事件。先通过 mMediaRecorder01.stop()停止录音，然后将录音文件名给 Adapter。具体代码如下所示：

```
/* 停止 */
myButton2.setOnClickListener(new ImageButton.OnClickListener()
{
  @Override
  public void onClick(View arg0)
  {
    // TODO Auto-generated method stub
```

```
        if (myRecAudioFile != null)
        {
          /* 停止录音 */
          mMediaRecorder01.stop();
          mMediaRecorder01.release();
          mMediaRecorder01 = null;
          /* 将录音文件名给 Adapter */
          adapter.add(myRecAudioFile.getName());
          myTextView1.setText("停止: " + myRecAudioFile.getName());
          myButton2.setEnabled(false);
          isStopRecord = true;
        }
      }
    });
```

(4) 设置单击播放按钮后的处理事件，单击后将打开播放程序。具体代码如下所示：

```
    /* 播放 */
    myButton3.setOnClickListener(new ImageButton.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
        // TODO Auto-generated method stub
        if (myPlayFile != null && myPlayFile.exists())
        {
          /* 打开播放程序 */
          openFile(myPlayFile);
        }
      }
    });
```

(5) 设置单击删除按钮后的处理事件。先将 Adapter 删除文件名，然后删除存在的录音文件。具体代码如下所示：

```
    /* 删除 */
    myButton4.setOnClickListener(new ImageButton.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
        // TODO Auto-generated method stub
        if (myPlayFile != null)
        {
          /* 先将 Adapter 删除文件名 */
          adapter.remove(myPlayFile.getName());
          /* 删除文件 */
          if (myPlayFile.exists())
            myPlayFile.delete();
          myTextView1.setText("完成删除");
        }
      }
    });
```

(6) 设置单击列表中文件的处理事件，当有文件单击后将删除及播放按钮设置为 Enable，并输出选择提示语句。具体代码如下所示：

```
  myListView1.setOnItemClickListener
  (new AdapterView.OnItemClickListener()
  {
    @Override
    public void onItemClick(AdapterView<?> arg0, View arg1,
        int arg2, long arg3)
    {
      /* 当有点击后将删除及播放按钮设置为 Enable */
      myButton3.setEnabled(true);
      myButton4.setEnabled(true);
      myPlayFile = new File(myRecAudioDir.getAbsolutePath()
          + File.separator
          + ((CheckedTextView) arg1).getText());
      myTextView1.setText("你选的是："
          + ((CheckedTextView) arg1).getText());
    }
  });
}
```

(7) 定义方法 onStop()，用于停止录音处理。具体代码如下所示：

```
@Override
protected void onStop()
{
  if (mMediaRecorder01 != null && !isStopRecord)
  {
    /* 停止录音 */
    mMediaRecorder01.stop();
    mMediaRecorder01.release();
    mMediaRecorder01 = null;
  }
  super.onStop();
}
```

(8) 定义方法 getRecordFiles()，用于获取文件的长度，并设置只获取“.amr”格式的文件。具体代码如下所示：

```
private void getRecordFiles()
{
  recordFiles = new ArrayList<String>();
  if (sdCardExit)
  {
    File files[] = myRecAudioDir.listFiles();
    if (files != null)
    {
      for (int i = 0; i < files.length; i++)
      {
        if (files[i].getName().indexOf(".") >= 0)
        {
          /* 只取.amr 文件 */
```

```
            String fileS = files[i].getName().substring(
                files[i].getName().indexOf("."));
            if (fileS.toLowerCase().equals(".amr"))
              recordFiles.add(files[i].getName());
          }
        }
      }
    }
  }
```

(9) 定义方法 openFile(File f)，用于打开播放指定的录音文件。具体代码如下所示：

```
/* 打开播放录音文件的程序 */
private void openFile(File f)
{
  Intent intent = new Intent();
  intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
  intent.setAction(android.content.Intent.ACTION_VIEW);
  String type = getMIMEType(f);
  intent.setDataAndType(Uri.fromFile(f), type);
  startActivity(intent);
}
```

(10) 定义方法 getMIMEType(File f)，用于文件的类型，在此设置了 audio、image 和其他类型共三大类。具体代码如下所示：

```
private String getMIMEType(File f)
{
  String end = f.getName().substring(
      f.getName().lastIndexOf(".") + 1, f.getName().length())
      .toLowerCase();
  String type = "";
  if (end.equals("mp3") || end.equals("aac")
      || end.equals("amr") || end.equals("mpeg")
      || end.equals("mp4"))
  {
    type = "audio";
  }
  else if (end.equals("jpg") || end.equals("gif")
      || end.equals("png") || end.equals("jpeg"))
  {
    type = "image";
  }
  else
  {
    type = "*";
  }
  type += "/*";
  return type;
}
}
```

在文件 AndroidManifest.xml 中打开录音权限。具体代码如下所示：

```
<uses-permission android:name="android.permission.RECORD_AUDIO">
```

至此整个展示结束，执行后的初始效果如图 12-13 所示。当单击“录音”按钮时开始录音处理，如图 12-14 所示。当单击“停止”按钮后停止录音处理，并在列表中显示录制的音频文件，如图 12-15 所示；当选中音频文件，单击“删除”按钮后会删除选中音频文件，如图 12-16 所示；单击“播放”按钮后会播放选中的音频文件，如图 12-17 所示。



图 12-13　初始效果　　图 12-14　正在录音

图 12-15　显示录音的文件　　图 12-16　删除音频文件



图 12-17　播放音频文件

12.9　3gp 视频播放器

题　目	目　的	源码路径
题目 8	在 Android 手机中实现 3gp 视频播放	“光盘:\daima\12\example11”文件夹

12.9.1　我的想法

这个题目很棘手，只好向秘籍请教，在《Android SDK 4.0 密录》中写道：Android 中内置了 VideoView Widget 作为多媒体视频播放器，它可以浏览视频。VideoView Widget 与前面介绍

的 Widget 私用方法类似，必须先在 Layout XML 中定义 VideoView 属性，在程序中通过 findViewById()方法即可创建 VideoView 对象。

我决定预先准备两个.3gp 格式的视频文件上传到虚拟 SD 卡中，然后插入两个按钮，当单击按钮后分别实现对这两个视频文件的播放。

12.9.2 具体实现

编写的主程序文件是 example11.java，其具体实现代码如下。

(1) 设置是否安装存储卡 flag 的默认值为 false，然后设置全屏幕显示。具体代码如下所示：

```
/* 默认判别是否安装存储卡 flag 为 false */
private boolean bIfSDExist = false;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);

  /* 全屏幕 */
  getWindow().setFormat(PixelFormat.TRANSLUCENT);
  setContentView(R.layout.main);
```

(2) 判断存储卡是否存在，不存在则通过 mMakeTextToast 输出提示语句。具体代码如下所示：

```
  /* 判断存储卡是否存在 */
  if(android.os.Environment.getExternalStorageState().equals
  (android.os.Environment.MEDIA MOUNTED))
  {
    bIfSDExist = true;
  }
  else
  {
    bIfSDExist = false;
    mMakeTextToast
    (
      getResources().getText(R.string.str err nosd).toString(),
      true
    );
  }
```

(3) 定义单击处理事件，通过 playVideo(strVideoPath)播放第一个影片 1。具体代码如下所示：

```
  mButton01.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View arg0)
    {
```

```
        // TODO Auto-generated method stub
        if(bIfSDExist)
        {
          /* 播放影片路径 1 */
          strVideoPath = "file:///sdcard/hello.3gp";
          playVideo(strVideoPath);
        }
      }
    });
```

(4)　定义单击处理事件，通过 playVideo(strVideoPath)播放第二个影片 2。具体代码如下所示：

```
    mButton02.setOnClickListener(new Button.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
        // TODO Auto-generated method stub
        if(bIfSDExist)
        {
          /* 播放影片路径 2 */
          strVideoPath = "file:///sdcard/test.3gp";
          playVideo(strVideoPath);
        }
      }
    });
  }
```

(5)　自定义 VideoView 方法，用于播放指定路径的影片。具体代码如下所示：

```
  /* 自定义以 VideoView 播放影片 */
  private void playVideo(String strPath)
  {
    if(strPath!="")
    {
      /* 调用 VideoURI 方法，指定解析路径 */
      mVideoView01.setVideoURI(Uri.parse(strPath));

      /* 设置控制 Bar 显示于此 Context 中 */
      mVideoView01.setMediaController
      (new MediaController(example11.this));

      mVideoView01.requestFocus();

      /* 调用 VideoView.start()自动播放 */
      mVideoView01.start();
      if(mVideoView01.isPlaying())
      {
        /* 程序不会被运行，因 start()后尚需要 preparing() */
        mTextView01.setText("Now Playing:"+strPath);
```




```
        Log.i(TAG, strPath);
      }
    }
  }
```

(6) 定义 mMakeTextToast(String str, boolean isLong)方法，用于输出提醒语句。具体代码如下所示：

```
public void mMakeTextToast(String str, boolean isLong)
{
  if(isLong==true)
  {
    Toast.makeText(example11.this, str, Toast.LENGTH_LONG).show();
  }
  else
  {
    Toast.makeText(example11.this, str, Toast.LENGTH_SHORT).show();
  }
}
}
```

至此整个展示结束，当单击“播放 SD 3gp 影片 1”和“播放 SD 3gp 影片 2”按钮后分别播放预设的影片，执行效果如图 12-18 所示。



图 12-18　执行效果

12.10　铃 声 设 置

题　目	目　的	源码路径
题目 9	在 Android 手机中设置指定铃声	“光盘:\daima\12\example12”文件夹

12.10.1　我的想法

这个题目也很棘手，翻开《Android SDK 4.0 密录》后找到了答案：在 Android 中，通过

RingtoneManager 类来专门控制各种铃声。例如，常见的来电铃声、闹钟铃声、警告、信息通知等。RingtoneManager 类的常用方法如下。

- getActualDefaultRingtoneUri：获取指定类型的当前默认铃声。
- getCursor：返回所有可用铃声的游标。
- getDefaultType：获取指定 URL 默认的铃声类型。
- getDefaultUri：返回指定类型默认铃声的 URL。
- getRingtoneUri：返回指定位置铃声的 URL。
- getRingtonePosition：获取指定铃声的位置。
- getValidRingtoneUri：获取一个可用铃声的位置。
- isDefault：获取指定 URL 是否是默认的铃声。
- setActualDefaultRingtoneUri：设置默认的铃声。

在 Android 系统中，默认的铃声存储在“system/medio/audio”中，而下载的铃声一般被保存在 SD 卡中，在此假设下载的铃声分别保存在 SD 卡的以下目录中。

- sdcard/music/ringtone：一般来电铃声。
- sdcard/music/alarm：闹钟铃声。
- sdcard/music/notification：警告、通知铃声。

12.10.2　具体实现

(1) 设置单击按钮 mButtonRingtone 后的处理事件，打开系统铃声设置后进行铃声设置。具体代码如下所示：

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mButtonRingtone = (Button) findViewById(R.id.ButtonRingtone);
    mButtonAlarm = (Button) findViewById(R.id.ButtonAlarm);
    mButtonNotification = (Button) findViewById(R.id.ButtonNotification);
    /* 设置来电铃声 */
    mButtonRingtone.setOnClickListener(new Button.OnClickListener()
    {
        @Override
        public void onClick(View arg0)
        {
            if (bFolder(strRingtoneFolder))
            {
                //打开系统铃声设置
                Intent intent = new Intent(RingtoneManager.ACTION_RINGTONE_PICKER);
                //类型为来电 RINGTONE
                intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE,
                RingtoneManager.TYPE_RINGTONE);
                //设置显示的 title
```

```
                    intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TITLE, "设
                    置来电铃声");
                    //当设置完成之后返回到当前的 Activity
                    startActivityForResult(intent, ButtonRingtone);
                }
            }
        });
```

(2) 设置单击按钮 mButtonAlarm 后的处理事件，打开系统铃声设置后进行铃声设置。具体代码如下所示：

```
        /* 设置闹钟铃声 */
        mButtonAlarm.setOnClickListener(new Button.OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                if (bFolder(strAlarmFolder))
                {
                    //打开系统铃声设置
                    Intent intent = new Intent(RingtoneManager.ACTION_RINGTONE_PICKER);
                    //设置铃声类型和 title
                    intent.putExtra(RingtoneManager.EXTRA RINGTONE TYPE,
                    RingtoneManager.TYPE_ALARM);
                    intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TITLE, "设
                    置闹铃铃声");
                    //当设置完成之后返回到当前的 Activity
                    startActivityForResult(intent, ButtonAlarm);
                }
            }
        });
```

(3) 设置单击按钮 mButtonNotification 后的处理事件，打开系统铃声设置后进行铃声设置。具体代码如下所示：

```
        /* 设置通知铃声 */
        mButtonNotification.setOnClickListener(new Button.OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                if (bFolder(strNotificationFolder))
                {
                    //打开系统铃声设置
                    Intent intent = new Intent(RingtoneManager.ACTION_RINGTONE_PICKER);
                    //设置铃声类型和 title
                    intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE,
                    RingtoneManager.TYPE_NOTIFICATION);
                    intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TITLE, "设
                    置通知铃声");
                    //当设置完成之后返回到当前的 Activity
```

```
                startActivityForResult(intent, ButtonNotification);
            }
        }
    });
}
```

(4) 定义方法 onActivityResult，用于设置铃声之后的回调函数。具体代码如下所示：

```
/* 当设置铃声之后的回调函数 */
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    // TODO Auto-generated method stub
    if (resultCode != RESULT_OK)
    {
        return;
    }
    switch (requestCode)
    {
        case ButtonRingtone:
            try
            {
                //得到我们选择的铃声
                Uri pickedUri = data.getParcelableExtra(RingtoneManager.
                EXTRA_RINGTONE_PICKED_URI);
                //将我们选择的铃声设置成为默认
                if (pickedUri != null)
                {
                    RingtoneManager.setActualDefaultRingtoneUri(Activity01.this,
                    RingtoneManager.TYPE_RINGTONE, pickedUri);
                }
            }
            catch (Exception e)
            {
            }
            break;
        case ButtonAlarm:
            try
            {
                //得到我们选择的铃声
                Uri pickedUri = data.getParcelableExtra(RingtoneManager.
                EXTRA_RINGTONE_PICKED_URI);
                //将我们选择的铃声设置成为默认
                if (pickedUri != null)
                {

RingtoneManager.setActualDefaultRingtoneUri(Activity01.this,
RingtoneManager.TYPE_ALARM, pickedUri);
                }
            }
            catch (Exception e)
```

```
                {
                }
                break;
            case ButtonNotification:
                try
                {
                    //得到我们选择的铃声
                    Uri pickedUri = data.getParcelableExtra(RingtoneManager.
                    EXTRA_RINGTONE_PICKED_URI);
                    //将我们选择的铃声设置成为默认
                    if (pickedUri != null)
                    {

    RingtoneManager.setActualDefaultRingtoneUri(Activity01.this,
    RingtoneManager.TYPE_NOTIFICATION, pickedUri);
                    }
                }
                catch (Exception e)
                {
                }
                break;
        }
        super.onActivityResult(requestCode, resultCode, data);
    }
```

(5) 定义方法 boolean bFolder()，用于检测是否存在指定的文件夹，如果不存在则进行创建。具体代码如下所示：

```
    private boolean bFolder(String strFolder)
    {
        boolean btmp = false;
        File f = new File(strFolder);
        if (!f.exists())
        {
            if (f.mkdirs())
            {
                btmp = true;
            }
            else
            {
                btmp = false;
            }
        }
        else
        {
            btmp = true;
        }
        return btmp;
    }
}
```

执行后可以分别设置三种类型的铃声，执行效果如图 12-19 所示。



图 12-19　执行效果



第 13 章　第四关：千里传音

在移动手机应用中，Internet 是一个重要的构成模块。现在的智能手机逐渐发展为更加灵活的个人移动电脑设备。在本节的内容中，将通过几个典型实例的实现过程来详细介绍 Android 在 Internet 领域应用的基本知识，并讲解各项技术的具体使用流程。

13.1　循 序 渐 进

学习 Android 开发需要循序渐进，不能幻想像虚竹那样突然获得一甲子功力的美事。对于 Android 新手来说，打好基础尤为重要。每一个知识点、每一个语法知识和每一个实践演练都要认真对待。都说习武之人如果内力达到一定造诣可以千里传音，具体真假不得而知。但是当今科学技术的进步，通过网络可以和各地的朋友进行交流。接下来的第四关——千里传音，本关的所有题目同网络有关，希望读者认真体会每一个实例的实现方法和技巧，为以后的开发之路打好基础。

13.2　实现网页浏览

题　目	目　的	源码路径
题目 1	在 Android 中编程实现网页浏览	“光盘:\daima\13\example2”文件夹

13.2.1　我的想法

对于现在的手机来说，实现网上冲浪已经不是什么难事，看似复杂其实实现起来很简单。因为在 Android 中内置了一个 WebKit 引擎，里面的 WebView Widget 能够迅速实现网页浏览，我们可以通过 WebView.loadUrl 来加载网址，所以从 EditText 中传入要浏览的网址后，就可以在 WebView 中加载网页的内容了。

13.2.2 具体实现

编写的主程序文件是 example2.java，通过 setOnClickListener 监听按钮单击事件，单击“箭头”按钮后先抓取 EditText 中的数据，然后打开此网址并在 WebView 中显示网页内容。主要代码如下所示：

```
…

    /*当单击箭头后*/
    mImageButton1.setOnClickListener(new
                                ImageButton.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
        // TODO Auto-generated method stub
        {
          mImageButton1.setImageResource(R.drawable.go_2);
          /*抓取 EditText 中的数据*/
          String strURI = (mEditText1.getText().toString());
          /*  WebView 显示网页内容龤*/
          mWebView1.loadUrl(strURI);
          Toast.makeText(
              example2.this,getString(R.string.load)+strURI,
                    Toast.LENGTH_LONG)
              .show();
        }
      }
    });
  }
}
```

至此整个展示结束，执行后显示一个文本框，在此可以输入网址，如图 13-1 所示。输入网址并单击▶按钮后，将显示此网页的内容，如图 13-2 所示。

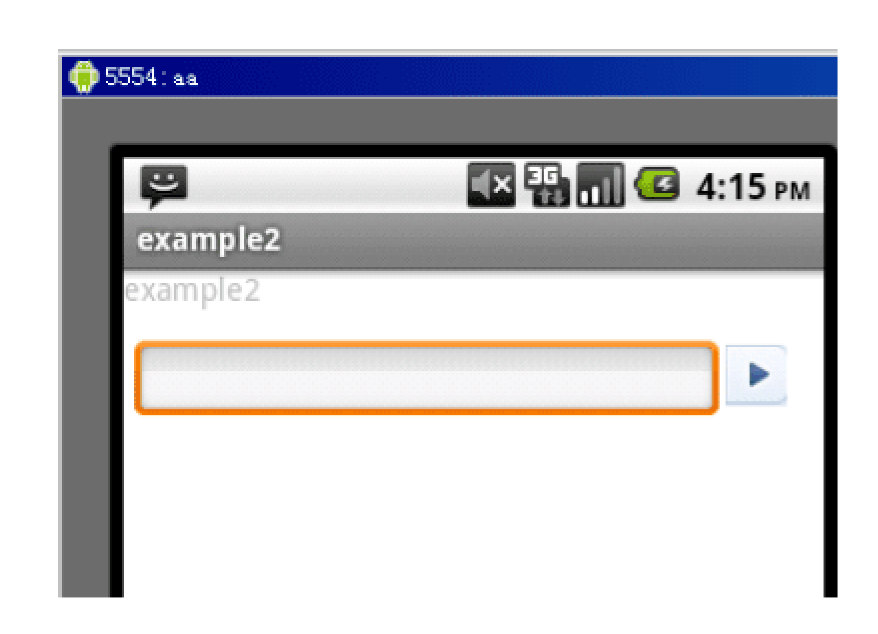


图 13-1　输入网址



图 13-2　打开的网页

13.3　使用 HTML 程序就是这么简单

题　目	目　的	源码路径
题目 2	在 Android 中使用 HTML 程序	“光盘:\daima\13\example3”文件夹

13.3.1　我的想法

WebView 是一个嵌入式的浏览器，可以直接使用方法 WebView.loadData()，将 HTML 标记传递给 WebView 对象，让 Android 手机程序变为 Web 浏览器。这样，网页程序就被放在 WebView 中运行，如同一个 Web Appliction。在当前移动程序中，网页下载和动画展示等都利用了 WebView 中的 loadData 来载入网页。

13.3.2　具体实现

编写的主程序文件是 example3.java，在 loadData 中插入了指定的 HTML 代码，通过 HTML 代码显示了一幅图片和文字，并且还插入了超级链接功能。具体代码如下所示：

```
public class example3 extends Activity
{
  private WebView mWebView1;
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mWebView1 = (WebView) findViewById(R.id.myWebView1);

    /*自行设置 WebView 要显示的网页内容*/
    mWebView1.
      loadData(
      "<html><body><p>aaaaaaa</p>" +
      "<div class='widget-content'> "+
      "<a href=http://www.sohu.com>" +
      "<img src=http://hiphotos.baidu.com/chaojihedan/pic/item/
      bbddf5efc260f133fdfa3cd13.jpg />" +
      "<a href=http://www.sohu.com>Link Blog</a>" +
      "</body></html>", "text/html", "utf-8");
    }
}
```

至此整个展示结束，执行后将显示 HTML 产生的页面，如图 13-3 所示。单击超链接后会来到指定的目标页面，如图 13-4 所示。

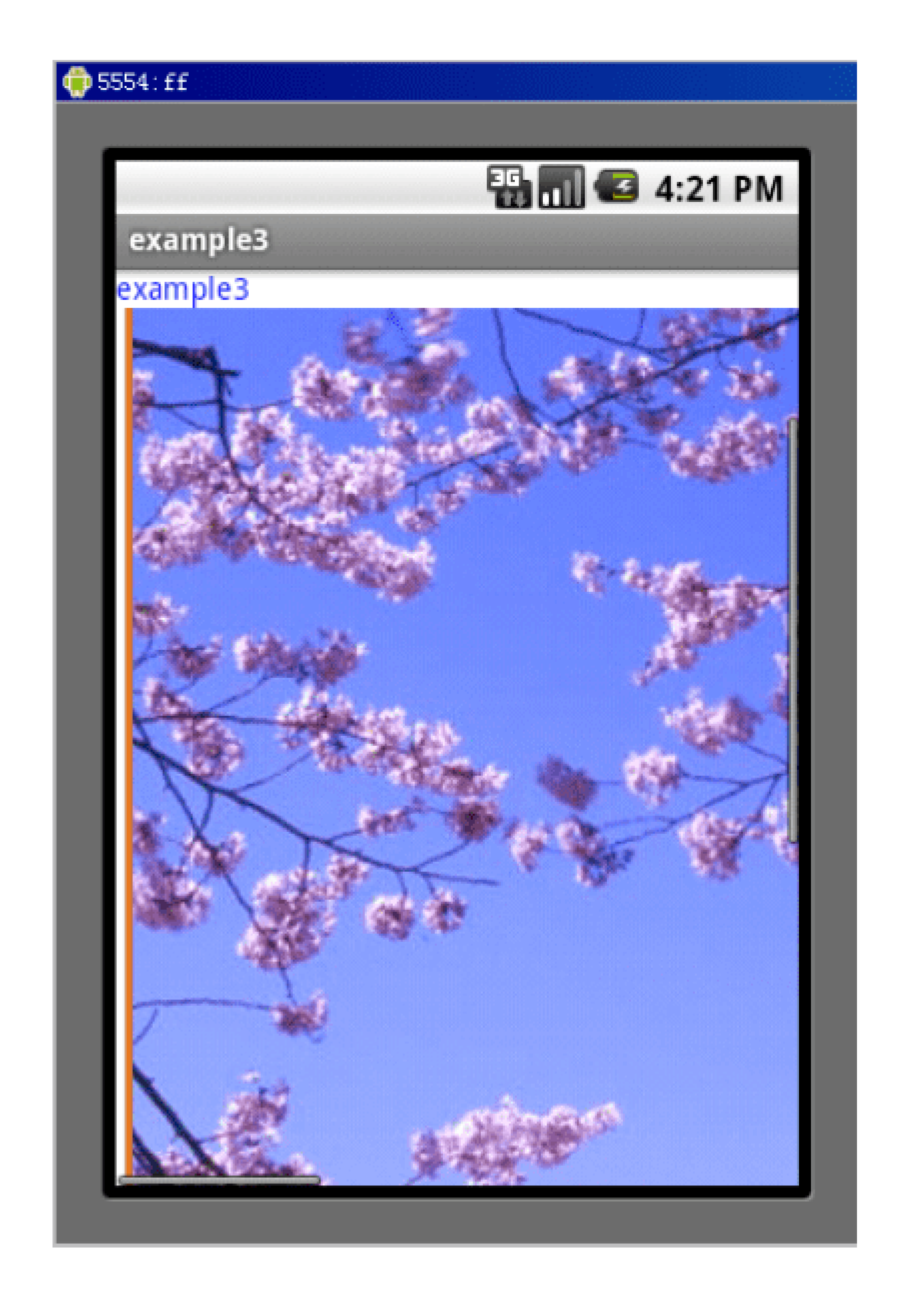



图 13-3　输入网址



图 13-4　目标页面

13.4　调用内置浏览器打开网页

题　目	目　的	源码路径
题目 3	调用 Android 内置浏览器打开网页	“光盘:\daima\13\example4”文件夹

13.4.1　我的想法

安卓有一个内置浏览器，我只需定义一个 ListView 在里面列表显示 4 个菜单，当单击菜单后会连接到指定的页面。当 ListView 的 ItemClick()事件发生时，通过 Intent(Intent.ACTION_VIEW，uri)来打开内置的浏览器并浏览 ListView 中创建的网页。

13.4.2　具体实现

编写的主程序文件是 example4.java，下面开始介绍其实现流程。

(1)　分别声明一个 ListView 和 TextView 对象变量，然后声明一个 String array 变量来存储收藏夹列表，最后声明一个 String 变量来存储网址。具体代码如下所示：

```
public class example4 extends Activity
{
  /*声明一个 ListView,TextView 对象变量
   * 一个 String array 变量存储收藏夹列表
   * 与 String 变量来存储网址*/
  private ListView mListView1;
```

```
private TextView mTextView1;
private String[] myFavor;
private String  myUrl;
```

(2) 先通过 findViewById 构造器创建 ListView 与 TextView 对象，将 string.xml 中的信息导入到列表中。具体代码如下所示：

```
/*通过 findViewById 构造器创建 ListView 与 TextView 对象*/
mListView1 =(ListView) findViewById(R.id.myListView1);
mTextView1 = (TextView) findViewById(R.id.myTextView1);
mTextView1.setText(getResources().getString(R.string.hello));
/*将列表通过 string.xml 中导入*/
myFavor = new String[] {
                      getResources().getString
                      (R.string.str_list_url1),
                      getResources().getString
                      (R.string.str_list_url2),
                      getResources().getString
                      (R.string.str_list_url3),
                      getResources().getString
                      (R.string.str_list_url4)
                    };
```

(3) 自定义一个 ArrayAdapter 准备传入到 ListView 中，并将 myFavor 列表以参数传入。然后自定义完成的 ArrayAdapter 传入自定义的 ListView 中，将 ListAdapter 的可选(Focusable)菜单选项打开，最后设置 ListView 选项的 nItemClickListener。具体代码如下所示：

```
/*自定义一 ArrayAdapter 准备传入 ListView 中，并将 myFavor 列表以参数传入*/
ArrayAdapter<String> adapter = new
ArrayAdapter<String>
(example4.this, android.R.layout.simple_list_item_1, myFavor);

/*将自定义完成的 ArrayAdapter 传入自定义的 ListView 中*/
mListView1.setAdapter(adapter);
/*将 ListAdapter 的可选(Focusable)菜单选项打开*/
mListView1.setItemsCanFocus(true);
/*设置 ListView 菜单选项设为每次只能单一选项*/
mListView1.setChoiceMode
(ListView.CHOICE_MODE_SINGLE);
/*设置 ListView 选项的 nItemClickListener*/
mListView1.setOnItemClickListener
(new ListView.OnItemClickListener()
{
```

(4) 定义覆盖 onItemClick 方法，当用户单击一个 Item 后会进行比较，并从 string.xml 中取出对应的 URL 网址，将字符串转换为 URL 对象。具体代码如下所示：

```
/*覆盖 OnItemClick 方法*/
public void onItemClick
(AdapterView<?> arg0, View arg1, int arg2,long arg3)
{
  // TODO Auto-generated method stub
  /*若所选菜单的文字与 myFavor 字符串数组第一个文字相同*/
```

```
        if(arg0.getAdapter().getItem(arg2).toString()==
        myFavor[0].toString())
        {
          /*取得网址并调用 goToUrl()方法*/
          myUrl=getResources().getString(R.string.str_url1);
          goToUrl(myUrl);
        }
        /*若所选菜单的文字与 myFavor 字符串数组第二个文字相同*/
        else if (arg0.getAdapter().getItem(arg2).toString()==
        myFavor[1].toString())
        {
          /*取得网址并调用 goToUrl()方法*/
          myUrl=getResources().getString(R.string.str url2);
          goToUrl(myUrl);
        }
        /*若所选菜单的文字与 myFavor 字符串数组第三个文字相同*/
        else if (arg0.getAdapter().getItem(arg2).toString()==
        myFavor[2].toString())
        {
          /*取得网址并调用 goToUrl()方法*/
          myUrl=getResources().getString(R.string.str_url3);
          goToUrl(myUrl);
        }
        /*若所选菜单的文字与 myFavor 字符串数组第四个文字相同*/
        else if (arg0.getAdapter().getItem(arg2).toString()==
        myFavor[3].toString())
        {
          /*取得网址并调用 goToUrl()方法*/
          myUrl=getResources().getString(R.string.str_url4);
          goToUrl(myUrl);
        }
        /*以上皆非*/
        else
        {
          /*显示错误信息*/
          mTextView1.setText("Ooops!!出错了");
        }
      }
    });
  }
```

(5) 定义方法 goToUrl(String url)用于打开网址为 URL 的网页。具体代码如下所示：

```
  /*打开网页的方法*/
  private void goToUrl(String url)
  {
    Uri uri = Uri.parse(url);
    Intent intent = new Intent(Intent.ACTION_VIEW, uri);
    startActivity(intent);
  }
}
```

至此整个展示结束，执行后将列表显示 4 个菜单项，如图 13-5 所示。当单击一个菜单项后，

会来到对应的目标页面，如图 13-6 所示。

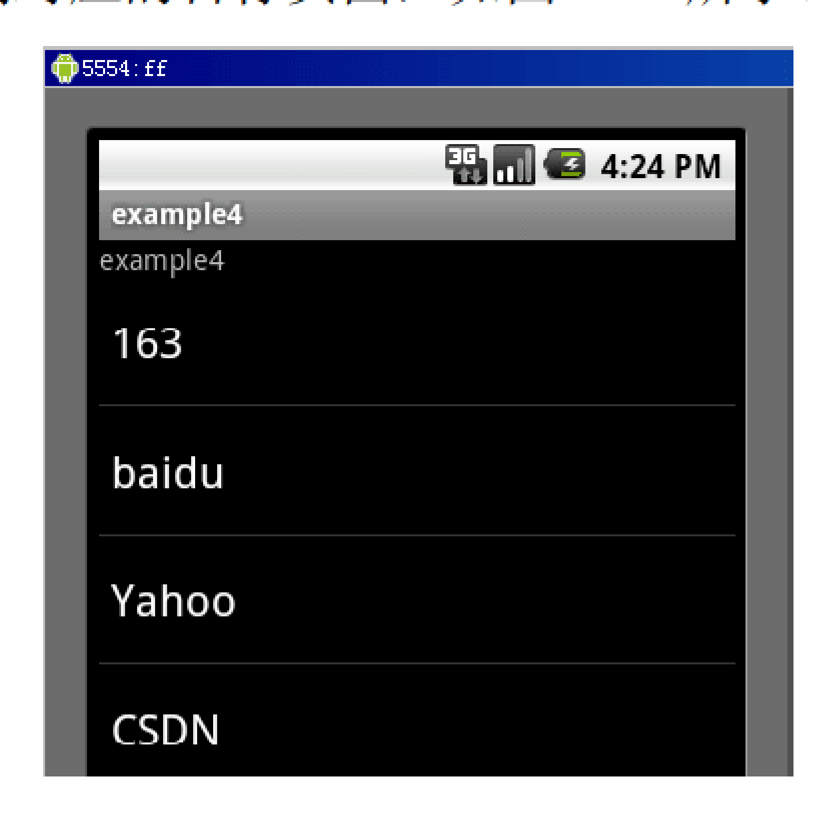


图 13-5　4 个菜单项



图 13-6　打开的网页

13.5　Gallery 中显示 QQ 空间的照片

题　目	目　的	源码路径
题目 4	用 Gallery 调用 QQ 空间照片并显示	“光盘:\daima\13\example5”文件夹

13.5.1　我的想法

这个题目很时髦，我不知如何下手，只好从《Android SDK 4.0 密录》中获取答案，密录中写道：网络真是无奇不有，在 QQ 空间中我们可以存放照片，不需要在 Gallery 中存放照片了，可以直接从网络中调用存放的照片并在 Gallery 中显示出来，这样可以节约手机的存储空间。

原来如此，我需要将 URL 网址的相片实时处理下载后，将 InputStream 转换为 Bitmap，这样才能放入 BaseAdapter 中取用。在运行之前，需要预先准备照片并上传到网络空间中，获取相片的连接后再以 String 数组方式放到程序中，对 BaseAdapter 稍作修改，配合 URL 对象的访问和 URLConnection 连接的处理。

13.5.2　具体实现

编写的主程序文件是 example5.java，下面开始介绍其实现流程。

(1)　分别声明 Gallery 中要显示的 5 幅图片的地址栏字符串。具体代码如下所示：

```
public class example5 extends Activity
{
  private Gallery myGallery01;
  /* 地址栏字符串 */
  private String[] myImageURL = new String[]
  {
     "http://b213.photo.store.qq.com/http_imgload.cgi?/"
         +
"rurl4_b=086a67cbd6a8cfb4389ea2b48efab6f322f755a085107a7aeeaa56fc1358b1bd124
```

```
186254e021f0655732688e69f060725491f8ae82e8e5508dbe9821670e2baf04e92dedc97e3b
bf28e5605596aa991c13220f1&a=27&b=27",
     "http://b213.photo.store.qq.com/http_imgload.cgi?/"
        +
"rurl4_b=086a67cbd6a8cfb4389ea2b48efab6f3ea78f5797abbbaa617259f2d2a980a5468f
2801897cfcc2b78af92fbb87565ed7a3a08041daff2dd9ccd26d3cc6198e41f2d205c8a0c445
325771e8a179215999afaf9f3&a=27&b=27",
     "http://b213.photo.store.qq.com/http_imgload.cgi?/"
        +
"rurl4_b=2a9dcf1fd909a7ed3ce8951f738608982f26d812b3a5fc96e221b85fc085e7cc326
4ee20730f0fd3a1f7aca06740db7a6153d9357467ca39f82b866b6fbe3cd94bbdd10ed01841e
67c95d8e4af8890b7ced40869&a=30&b=27",
     "http://b213.photo.store.qq.com/http imgload.cgi?/"
        +
"rurl4_b=2a9dcf1fd909a7ed3ce8951f73860898bb7ff57a8cb7747c9f0eb6a02124850b709
c0b86f086a4ba5653eeb71dd4b01e4a58f407e2eec9433cd8d4bc0b88fda56260c2c8beb34eb
ab77b610c7131393f82e774ef&a=27&b=27",
     "http://b213.photo.store.qq.com/http_imgload.cgi?/"
        +
"rurl4_b=2a9dcf1fd909a7ed3ce8951f73860898158d252489f84e7d2a83d44c01b7bb12b2c
19ca0efdd555dba788407fd01e9de45524b11a9793f532624197bc8d14c84ae78ddebafe4357
e4eedc60e9e510224367490bf&a=27&b=27" };
```

(2) 引入布局文件 main.xml，定义类成员 myContext Context 对象，然后设置只有一个参数 C 的构造器，即要存储的 Context。先获取 Gallery 属性的 Index id，设置对象的 styleable 属性能够反复使用。具体代码如下所示：

```
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  myGallery01 = (Gallery) findViewById(R.id.myGallery01);
  myGallery01.setAdapter(new myInternetGalleryAdapter(this));
}

/* 用 BaseAdapter */
public class myInternetGalleryAdapter extends BaseAdapter
{
  /* 类成员 myContext Context 对象 */
  private Context myContext;
  private int mGalleryItemBackground;
  /*构造器只有一个参数，即要存储的 Context */
  public myInternetGalleryAdapter(Context c)
  {
    this.myContext = c;
    TypedArray a = myContext
        .obtainStyledAttributes(R.styleable.Gallery);

    /*  获取 Gallery 属性的 Index id */
    mGalleryItemBackground = a.getResourceId(
    R.styleable.Gallery android galleryItemBackground, 0);
    /* 把对象的 styleable 属性能够反复使用 */
```

```
    a.recycle();
  }
```

(3) 定义方法 getCount()，用于返回全部已定义图片的总量，然后定义 getItem(int position)，用于使用 getItem 方法获取当前容器中图像数的数组 ID。具体代码如下所示：

```
/*  返回全部已定义图片的总量 */
public int getCount()
{
  return myImageURL.length;
}
/* 使用 getItem 方法获取当前容器中图像数的数组 ID */
public Object getItem(int position)
{
  return position;
}
public long getItemId(int position)
{
  return position;
}
```

(4) 定义 getScale(boolean focused, int offset)，能够根据中央位移量，利用 getScale 返回 Views 的大小(0.0f to 1.0f)。具体代码如下所示：

```
/* 根据中央位移量，利用 getScale 返回 views 的大小(0.0f to 1.0f) */
public float getScale(boolean focused, int offset)
{
  /* Formula: 1 / (2 ^ offset) */
  return Math.max(0, 1.0f / (float) Math.pow(2, Math
      .abs(offset)));
}
```

(5) 定义 getView，能够根据中央位移量，用于获取当前要显示的图像 View，传入数组 ID 值使之读取并成像处理。具体流程如下。

第 1 步：创建 ImageView 对象，并用 new URL 将对象网址传入，然后获取连接。

第 2 步：获取返回的 InputStream 并将 InputStream 变为 Bitmap，然后关闭 InputStream。

第 3 步：设置 Bitmap 到 ImageView 中，然后设置 ImageView 的宽和高，单位是 dip。

第 4 步：设置 Gallery 背景图。

具体代码如下所示：

```
@Override
public View getView(int position, View convertView,
    ViewGroup parent)
{
  // TODO Auto-generated method stub
  /* 创建 ImageView 对象*/

  ImageView imageView = new ImageView(this.myContext);
  try
  {
    /* new URL 将对象网址传入 */
```

```
            URL aryURI = new URL(myImageURL[position]);
            /*   获取连接 */
            URLConnection conn = aryURI.openConnection();
            conn.connect();
            /*   获取返回的 InputStream */
            InputStream is = conn.getInputStream();
            /* 将 InputStream 变为 Bitmap */
            Bitmap bm = BitmapFactory.decodeStream(is);
            /* 关闭 InputStream */
            is.close();
            /* 设置 Bitmap 到 ImageView 中 */
            imageView.setImageBitmap(bm);
          } catch (IOException e)
          {
            e.printStackTrace();
          }
          imageView.setScaleType(ImageView.ScaleType.FIT_XY);
          /* 设置 ImageView 的宽和高，单位是 dip */
          imageView.setLayoutParams(new Gallery.LayoutParams(200, 150));
          /* 设置 Gallery 背景图*/
          imageView.setBackgroundResource(mGalleryItemBackground);
          return imageView;
        }
      }
    }
```

至此整个展示结束，执行后将在 Gallery 中显示网络中的图片，如图 13-7 所示。

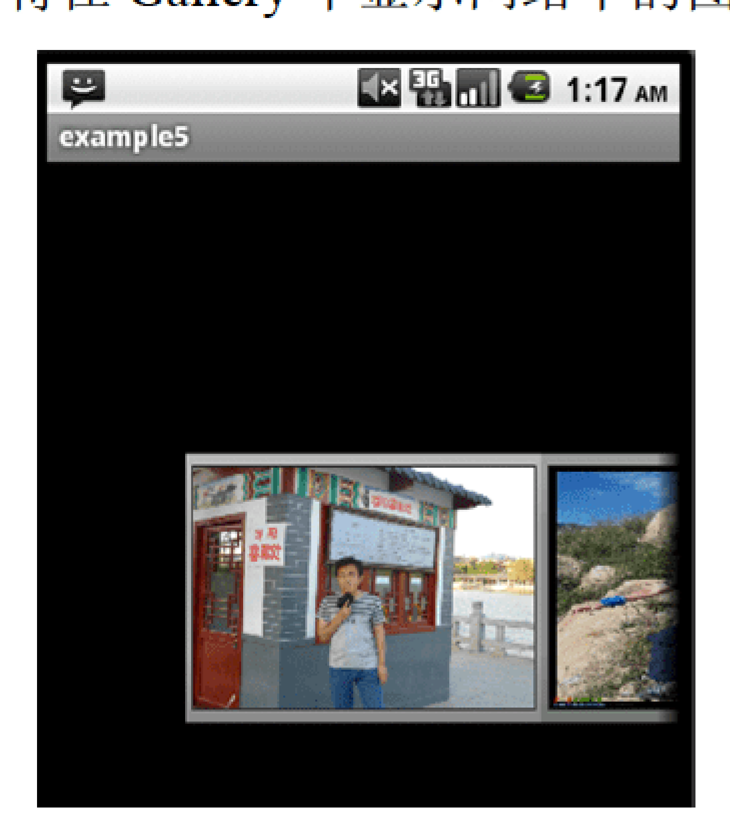


图 13-7　执行效果

13.6　播放网络 MP3

题　目	目　的	源码路径
题目 5	在 Android 中通过网络下载并播放 MP3	“光盘:\daima\13\example6”文件夹

13.6.1　我的想法

为了节约手机的存储空间，在听音乐时可以通过从网络中下载的方式来播放 MP3。方法很简单，先插入 4 个按钮，分别用于实现播放、暂停、重新播放和停止处理。执行后，通过 Runnable 发起运行线程，在线程中远程下载指定的 MP3 文件，这是通过网络传输方式下载的。下载完毕后，临时保存到 SD 卡中，这样可以通过 4 个按钮对音乐进行控制。当程序关闭后，删除 SD 卡中的临时性文件即可。

13.6.2　具体实现

编写的主程序文件是 example6.java，下面开始介绍其实现流程。

(1)　定义 currentFilePath，用于记录当前正在播放 MP3 的地址 URL，定义 currentTempFilePath 表示当前播放 MP3 的路径。具体代码如下所示：

```
/* 钍记录当前正在播放 MP3 的地址 URL      */
private String currentFilePath = "";

/*当前播放 MP3 的路径*/
private String currentTempFilePath = "";
private String strVideoURL = "";
```

(2)　先引入主布局文件 main.xml，然后通过 strVideoURL 设置要播放 MP3 文件的网址。并设置透明度。具体代码如下所示：

```
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
/* MP3 文件不会被下载到 local*/
strVideoURL="http://www.lrn.cn/zywh/xyyy/yyxs/200805/W020080505536315331311З.mp3";

mTextView01 = (TextView)findViewById(R.id.myTextView1);
/*设置透明度*/
getWindow().setFormat(PixelFormat.TRANSPARENT);
mPlay = (ImageButton)findViewById(R.id.play);
mReset = (ImageButton)findViewById(R.id.reset);
mPause = (ImageButton)findViewById(R.id.pause);
mStop = (ImageButton)findViewById(R.id.stop);
```

(3)　设置单击“播放”按钮所触发的处理事件。具体代码如下所示：

```
/* 播放按钮 */
mPlay.setOnClickListener(new ImageButton.OnClickListener()
{
  public void onClick(View view)
  {
    /* 调用播放影片 Function */
    playVideo(strVideoURL);
    mTextView01.setText
    (
      getResources().getText(R.string.str_play).toString()+
```

```
        "\n"+ strVideoURL
      );
    }
  });
```

(4) 设置单击“重新播放”按钮所触发的处理事件。具体代码如下所示：

```
/* 重新播放 */
mReset.setOnClickListener(new ImageButton.OnClickListener()
{
  public void onClick(View view)
  {
    if(bIsReleased == false)
    {
      if (mMediaPlayer01 != null)
      {
        mMediaPlayer01.seekTo(0);
        mTextView01.setText(R.string.str_play);
      }
    }
  }
});
```

(5) 设置单击“暂停播放”按钮所触发的处理事件。具体代码如下所示：

```
/* 暂停播放 */
mPause.setOnClickListener(new ImageButton.OnClickListener()
{
  public void onClick(View view)
  {
    if (mMediaPlayer01 != null)
    {
      if(bIsReleased == false)
      {
        if(bIsPaused==false)
        {
          mMediaPlayer01.pause();
          bIsPaused = true;
          mTextView01.setText(R.string.str pause);
        }
        else if(bIsPaused==true)
        {
          mMediaPlayer01.start();
          bIsPaused = false;
          mTextView01.setText(R.string.str_play);
        }
      }
    }
  }
});
```

(6) 设置单击“停止播放”按钮所触发的处理事件。具体代码如下所示：

```
    /* 停止 */
    mStop.setOnClickListener(new ImageButton.OnClickListener()
    {
      public void onClick(View view)
      {
        try
        {
          if (mMediaPlayer01 != null)
          {
            if(bIsReleased==false)
            {
              mMediaPlayer01.seekTo(0);
              mMediaPlayer01.pause();
              //mMediaPlayer01.stop();
              //mMediaPlayer01.release();
              //bIsReleased = true;
              mTextView01.setText(R.string.str_stop);
            }
          }
        }
        catch(Exception e)
        {
          mTextView01.setText(e.toString());
          Log.e(TAG, e.toString());
          e.printStackTrace();
        }
      }
    });
  }
```

(7) 定义方法 playVideo(final String strPath)，用于播放指定的 MP3，其播放的是存储卡中暂时保存的 MP3 文件。具体代码如下所示：

```
  private void playVideo(final String strPath)
  {
    try
    {
      if (strPath.equals(currentFilePath)&& mMediaPlayer01 != null)
      {
        mMediaPlayer01.start();
        return;
      }

      currentFilePath = strPath;

      mMediaPlayer01 = new MediaPlayer();
      mMediaPlayer01.setAudioStreamType(2);
```

(8) 定义 setOnErrorListener 实现错误处理。具体代码如下所示：

```
      /*错误事件 */
```

```
    mMediaPlayer01.setOnErrorListener(new MediaPlayer.OnErrorListener()
    {
      @Override
      public boolean onError(MediaPlayer mp, int what, int extra)
      {
        //TODO Auto-generated method stub
        Log.i(TAG, "Error on Listener, what: " + what + "extra: " + extra);
        return false;
      }
    });
```

(9) 定义 setOnBufferingUpdateListener，用于捕捉使用 MediaPlayer 缓冲区的更新事件。具体代码如下所示：

```
    /* 捕捉使用 MediaPlayer 缓冲区的更新事件 */
    mMediaPlayer01.setOnBufferingUpdateListener(new
MediaPlayer.OnBufferingUpdateListener()
    {
      @Override
      public void onBufferingUpdate(MediaPlayer mp, int percent)
      {
        //TODO Auto-generated method stub
        Log.i(TAG, "Update buffer: " + Integer.toString(percent)+ "%");
      }
    });
```

(10) 定义 setOnCompletionListener，用于实现播放完毕所触发的事件。具体代码如下所示：

```
    /* 播放完毕所触发的事件 */
    mMediaPlayer01.setOnCompletionListener(new
    MediaPlayer.OnCompletionListener()
    {
      @Override
      public void onCompletion(MediaPlayer mp)
      {
        //TODO Auto-generated method stub
        //delFile(currentTempFilePath);
        Log.i(TAG,"mMediaPlayer01 Listener Completed");
      }
    });
```

(11) 定义 setOnPreparedListener，用于开始阶段的监听 Listener。具体代码如下所示：

```
    /* 开始阶段的监听 Listener */
    mMediaPlayer01.setOnPreparedListener(new
    MediaPlayer.OnPreparedListener()
    {
      @Override
      public void onPrepared(MediaPlayer mp)
      {
        //TODO Auto-generated method stub
        Log.i(TAG,"Prepared Listener");
      }
    });
```

(12) 定义 Runnable 对象 r，用 Runnable 来确保文件在存储完毕后才开始 start()。先将文件存储到 SD 卡，然后在 setDataSource 后运行 prepare()，最后通过 mMediaPlayer01.start()开始播放 MP3。具体代码如下所示：

```
    /* 用 Runnable 来确保文件在存储完毕后才开始 start() */
    Runnable r = new Runnable()
    {
      public void run()
      {
        try
        {
          /* setDataSource 将文件存储到 SD 卡*/
          setDataSource(strPath);
          /*  因为线程顺利进行，所以在 setDataSource 后运行 prepare() */
          mMediaPlayer01.prepare();
          Log.i(TAG, "Duration: " + mMediaPlayer01.getDuration());
          /* 开始播放 MP3 */
          mMediaPlayer01.start();
          bIsReleased = false;
        }
        catch (Exception e)
        {
          Log.e(TAG, e.getMessage(), e);
        }
      }
    };
    new Thread(r).start();
  }
```

(13) 定义函数 setDataSource，用于存储 URL 的 MP3 文件到存储卡。首先判断传入的地址是否为 URL，然后创建 URL 对象和临时文件，当 fos 存储完毕后调用 MediaPlayer.setDataSource。具体代码如下所示：

```
/*  定义函数用于存储 URL 的 MP3 文件到存储卡  */
private void setDataSource(String strPath) throws Exception
{
  /*  判断传入的地址是否为 URL */
  if (!URLUtil.isNetworkUrl(strPath))
  {
    mMediaPlayer01.setDataSource(strPath);
  }
  else
  {
    if(bIsReleased == false)
    {
      /* 创建 URL 对象 */
      URL myURL = new URL(strPath);
      URLConnection conn = myURL.openConnection();
      conn.connect();
```

```
        /* 获取 URLConnection 的 InputStream */
        InputStream is = conn.getInputStream();
        if (is == null)
        {
          throw new RuntimeException("stream is null");
        }

        /* 创建临时文件  */
        File myTempFile = File.createTempFile("yinyue", "."+getFileExtension
        (strPath));
        currentTempFilePath = myTempFile.getAbsolutePath();
        /* currentTempFilePath = /sdcard/hippoplayertmp393213.mp3 */
        /*
        if(currentTempFilePath!="")
        {
          Log.i(TAG, currentTempFilePath);
        }
        */
        FileOutputStream fos = new FileOutputStream(myTempFile);
        byte buf[] = new byte[128];
        do
        {
          int numread = is.read(buf);
          if (numread <= 0)
          {
            break;
          }
          fos.write(buf, 0, numread);
        }while (true);

        /* 直到 fos 存储完毕，调用 MediaPlayer.setDataSource */
        mMediaPlayer01.setDataSource(currentTempFilePath);
        try
        {
          is.close();
        }
        catch (Exception ex)
        {
          Log.e(TAG, "error: " + ex.getMessage(), ex);
        }
      }
    }
  }
```

(14) 定义方法 getFileExtension(String strFileName)，用于获取音乐文件扩展名，如果无法顺利获取扩展名则默认为.dat。具体代码如下所示：

```
  /* 获取音乐文件扩展名自定义函数 */
  private String getFileExtension(String strFileName)
  {
```

```
    File myFile = new File(strFileName);
    String strFileExtension=myFile.getName();

strFileExtension=(strFileExtension.substring(strFileExtension.lastIndexOf(".
")+1)).toLowerCase();
    if(strFileExtension=="")
    {
      /* 如果无法顺利获取扩展名则默认为.dat */
      strFileExtension = "dat";
    }
    return strFileExtension;
  }
```

(15) 定义方法 delFile(String strFileName)，当离开程序时删除临时音乐文件。具体代码如下所示：

```
  /* 离开程序时需要调用自定义函数删除临时音乐文件*/
  private void delFile(String strFileName)
  {
    File myFile = new File(strFileName);
    if(myFile.exists())
    {
      myFile.delete();
    }
  }

  @Override
  protected void onPause()
  {
    //TODO Auto-generated method stub

    /*   删除临时文件镨 */
    try
    {
      delFile(currentTempFilePath);
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
    super.onPause();
  }
}
```

至此整个展示结束，执行后可以通过播放、暂停、重新播放和停止四个按钮控制指定 MP3 的处理，效果如图 13-8 所示。

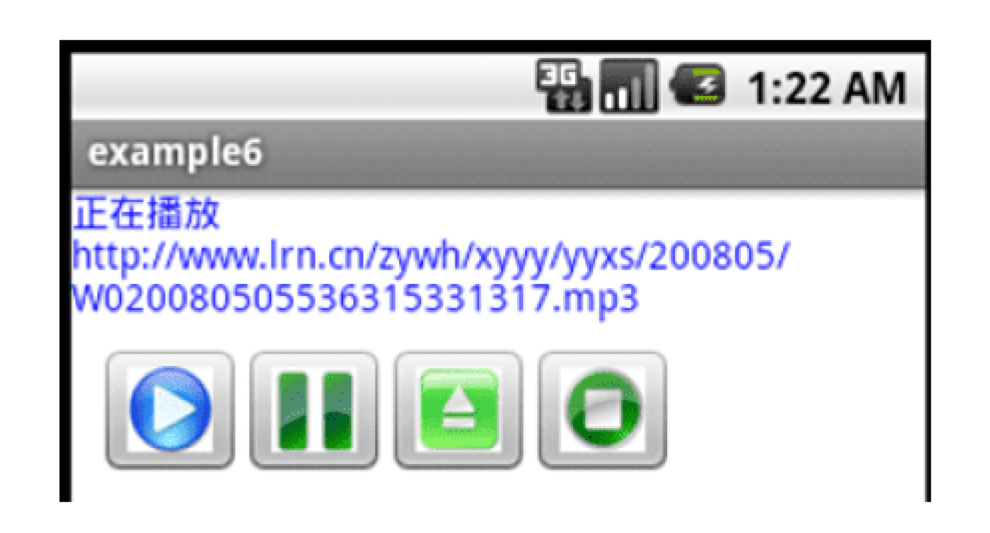


图 13-8 执行效果

13.7 远程下载手机铃声

题　目	目　的	源码路径
题目 6	下载网络中的音乐作为手机铃声	“光盘:\daima\13\example7”文件夹

13.7.1 我的想法

我知道铃声是移动手机的重要功能之一，我们可以从网络中直接下载 MP3 文件并设置为手机的铃声。这个题目师傅曾经考过我，我的解决方法为：通过 EditText 让用户输入一个指定的网址，当下载完成后打开 RingtoneManager.ACTION_ RINGTONE_PICKER，在打开 Intent 的同时传入一个参数，这个 ACTION_RINGTONE_PICKER 的 Intent 会带入刚才下载的文件让用户选择。在实现过程中，会判断下载文件是否完整，并判断用户是否已设置为铃声，下载后的铃声文件存储在哪里？在具体实现时，会以 SD 卡中的铃声文件作为存储网络下载音乐文件的路径，打开 RingtoneManager 的 ACTION_RINGTONE_PICKER 的 Intent 让用户找到下载的音乐，并设置为铃声。

13.7.2 具体实现

编写的主程序文件是 example7.java，其实现流程如下。

(1) 判断是否有/sdcard/music/ringtones 文件夹，不存在则输出提示。具体代码如下所示：

```
/*判断是否有/sdcard/music/ringtones 文件夹*/
if(bIfExistRingtoneFolder(strRingtoneFolder))
{
  Log.i(APP_TAG, "Ringtone Folder exists.");
}
```

(2) 在 String 中设置 strURL，通过 fileEx 和 getFile 取得文件的名称。具体代码如下所示：

```
mButton1.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View arg0)
  {
    // TODO Auto-generated method stub
```

```
      strURL = mEditText1.getText().toString();

       Toast.makeText(example13.this, getString(R.string.str_msg)
             ,Toast.LENGTH_SHORT).show();
      /*取得文件名称*/
      fileEx = strURL.substring(strURL.lastIndexOf(".")+1,strURL.
               length()).toLowerCase();
      fileNa = strURL.substring(strURL.lastIndexOf("/")+1,strURL.
               lastIndexOf("."));
      getFile(strURL);
    }
  });
}
```

(3) 定义方法 getMIMEType(File f)，用于判断文件 MimeType 的 method，首先取得扩展名，然后根据扩展名的类型决定 MimeType。具体代码如下所示：

```
/* 判断文件 MimeType 的 method */
private String getMIMEType(File f)
{
  String type="";
  String fName=f.getName();
  /* 取得扩展名 */
  String end=fName.substring(fName.lastIndexOf(".")+1,
                      fName.length()).toLowerCase();

  /* 依扩展名的类型决定 MimeType */
  if(end.equals("m4a")||end.equals("mp3")||end.equals("mid")||
      end.equals("xmf")||end.equals("ogg")||end.equals("wav"))
  {
    type = "audio";
  }
  else if(end.equals("3gp")||end.equals("mp4"))
  {
    type = "video";
  }
  else if(end.equals("jpg")||end.equals("gif")||
         end.equals("png")||end.equals("jpeg")||
          end.equals("bmp"))
  {
    type = "image";
  }
  else
  {
    type="*";
  }
  /*如果无法直接打开，就跳出软件列表给用户选择 */
  if(end.equals("image"))
  {
  }
  else
  {
```

```
    type += "/*";
  }
  return type;
}
```

(4) 定义方法 getFile(final String strPath)，用于获取最后的文件，如果地址和当前地址一样，则直接用 getDataSource 数据，如果有异常则输出异常信息。具体代码如下所示：

```
private void getFile(final String strPath)
{
  try
  {
    if (strPath.equals(currentFilePath) )
    {
     getDataSource(strPath);
    }
    currentFilePath = strPath;
    Runnable r = new Runnable()
    {
      public void run()
      {
        try
        {
          getDataSource(strPath);
        }
        catch (Exception e)
        {
          Log.e(APP_TAG, e.getMessage(), e);
        }
      }
    };
    new Thread(r).start();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}
```

(5) 定义方法 getDataSource(String strPath)，用于获取远程文件。如果地址错误，则输出错误信息。具体流程如下。

第 1 步：通过 myURL 获取 URL 并创建连接 conn。

第 2 步：通过 InputStream 下载文件并创建文件地址 myTempFile。

第 3 步：取得在内存盘的路径并将文件写入暂存盘。

具体代码如下所示：

```
/*取得远程文件*/
private void getDataSource(String strPath) throws Exception
{
  if (!URLUtil.isNetworkUrl(strPath))
  {
    mTextView1.setText("错误的 URL");
```

```
    }
    else
    {
      /*取得 URL*/
      URL myURL = new URL(strPath);
      /*创建连接*/
      URLConnection conn = myURL.openConnection();
      conn.connect();
      /*InputStream 下载文件*/
      InputStream is = conn.getInputStream();
      if (is == null)
      {
        throw new RuntimeException("stream is null");
      }
      /*创建文件地址*/
      File myTempFile = new File("/sdcard/music/ling/",
                     fileNa+"."+fileEx);
      /*取得在内存盘的路径*/
      currentTempFilePath = myTempFile.getAbsolutePath();
      /*将文件写入暂存盘*/
      FileOutputStream fos = new FileOutputStream(myTempFile);
      byte buf[] = new byte[128];
      do
      {
        int numread = is.read(buf);
        if (numread <= 0)
        {
          break;
        }
        fos.write(buf, 0, numread);
      }while (true);
```

(6) 打开 RingtonManager 进行铃声选择。通过 Intent 对象 intent 来设置铃声，然后设置显示铃声的文件夹和显示铃声开头，如果有异常则输出异常。具体代码如下所示：

```
      /* 打开 RingtonManager 进行铃声选择 */
      String uri = null;
      if(bIfExistRingtoneFolder(strRingtoneFolder))
      {
        /*设置铃声*/
        Intent intent = new Intent( RingtoneManager.
                  ACTION_RINGTONE_PICKER);
        /*设置显示铃声的文件夹*/
        intent.putExtra( RingtoneManager.EXTRA_RINGTONE_TYPE,
                  RingtoneManager.TYPE_RINGTONE);
        /*设置显示铃声开头*/
        intent.putExtra( RingtoneManager.EXTRA_RINGTONE_TITLE,
                  "设置铃声");
        if( uri != null)
        {
          intent.putExtra( RingtoneManager.
              EXTRA_RINGTONE_EXISTING_URI, Uri.parse( uri));
```

```
      }
      else
      {
        intent.putExtra( RingtoneManager.
              EXTRA_RINGTONE_EXISTING_URI,  (Uri)null);
      }
      startActivityForResult(intent, RINGTONE_PICKED);
    }
    try
    {
      is.close();
    }
    catch  (Exception ex)
    {
      Log.e(APP_TAG, "error: " + ex.getMessage(), ex);
    }
  }
}
```

(7) 定义方法 onActivityResult，能够根据用户选择的铃声设置保存对应的信息，当选择完毕后，会再次返回来选择 Activity。具体代码如下所示：

```
@Override
protected void onActivityResult(int requestCode,
                 int resultCode, Intent data)
{
  // TODO Auto-generated method stub
  if (resultCode != RESULT_OK)
  {
    return;
  }
  switch (requestCode)
  {
    case (RINGTONE_PICKED):
      try
      {
        Uri pickedUri = data.getParcelableExtra
        (RingtoneManager.EXTRA_RINGTONE_PICKED_URI);
        if(pickedUri!=null)
        {
          RingtoneManager.setActualDefaultRingtoneUri
          (example13.this,RingtoneManager.TYPE_RINGTONE,
              pickedUri);
        }
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
      break;
    default:
      break;
```

```
    }
    super.onActivityResult(requestCode, resultCode, data);
  }
```

(8)　定义 bIfExistRingtoneFolder，用于判断是否包含了“/sdcard/music/ringtones”文件夹。具体代码如下所示：

```
  /*判断是否/sdcard/music/ringtones 文件夹*/
  private boolean bIfExistRingtoneFolder(String strFolder)
  {
    boolean bReturn = false;

    File f = new File(strFolder);
    if(!f.exists())
    {
      /*创建/sdcard/music/ringtones 文件夹*/
      if(f.mkdirs())
      {
        bReturn = true;
      }
      else
      {
        bReturn = false;
      }
    }
    else
    {
      bReturn = true;
    }
    return bReturn;
  }
}
```

至此整个展示结束，执行后会先显示一个下载界面，如图 13-9 所示。单击“下载音乐”按钮后开始下载指定的 MP3 文件，下载完成后会弹出“设置铃声”对话框，如图 13-10 所示，选择一种铃声并单击 OK 按钮后完成铃声设置。

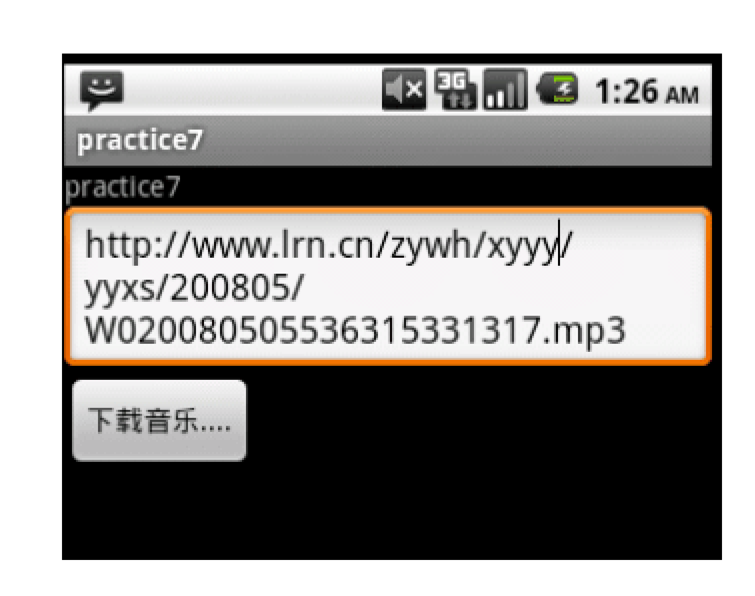


图 13-9　下载界面

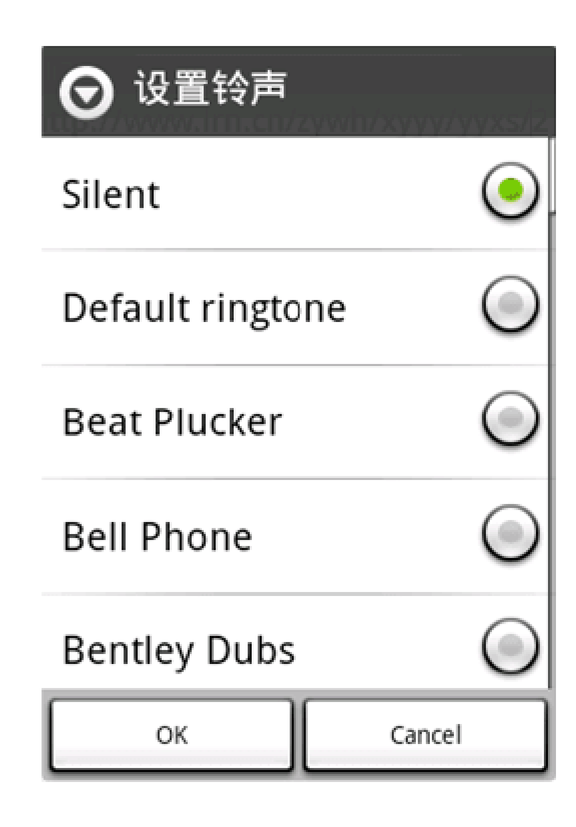


图 13-10　“设置铃声”对话框

13.8 文件上传至服务器

题　目	目　的	源码路径
题目 7	在 Android 中实现文件上传	“光盘:\daima\13\example9”文件夹

13.8.1 我的想法

文件上传功能对我来说并不陌生，难道在手机中同样可以实现类似网站的上传功能吗？为此，我专门研究了网站上传文件的实现原理和代码。在 Web 应用中，通常通过一个表单和上传程序来实现文件上传。具体的代码如下：

```
<FORM METHOD="POST" ACTION="do_upload.jsp" ENCTYPE="multipart/form-data">
<input type="FILE" name="FILE1" size="30">
<input type="submit" name="Submit" value="上传它! ">
</FORM>
```

上述代码是一个上传表单，供用户选择上传文件并通过文件 do_upload.jsp 实现上传处理。既然手机上传和网页上传类似，那么我也可以用 post 方式对服务器上的接收程序发出 request，触发该程序运行文件写入服务器的动作。在实现本实例前，需要搭建 Java 的 WEB SERVER 并在服务器上预先编写一个接收文件程序，在手机目录下准备要上传的资料，在此设置要上传的文件路径如下：

```
data/data/irdc.example9/image.jpg
```

都设置完成了就可以准备上传处理文件 upload.jsp 进行上传了。

13.8.2 具体实现

编写的主程序文件是 example9.java，其实现流程如下。

(1) 通过 mText1 获取文件路径，根据 mText2 设置上传网址，设置按钮的 onclick 事件处理并单击按钮后调用上传方法 uploadFile()。具体代码如下所示：

```
    mText1 = (TextView) findViewById(R.id.myText2);
    mText1.setText("文件路径: \n"+uploadFile);
    mText2 = (TextView) findViewById(R.id.myText3);
    mText2.setText("上传网址: \n"+actionUrl);
    /* 设置 mButton 的 onClick 事件处理 */
    mButton = (Button) findViewById(R.id.myButton);
    mButton.setOnClickListener(new View.OnClickListener()
    {
      public void onClick(View v)
      {
        uploadFile();
      }
    });
  }
```

(2) 定义方法 uploadFile()，用于将文件上传至 Server，具体流程如下。
第 1 步：设置传送的 method=POST。
第 2 步：设置 DataOutputStream 和获取文件的 FileInputStream。
第 3 步：设置每次写入 1024byte，然后从文件读取数据至缓冲区。
第 4 步：获取 Response 的内容并把 Response 在 Dialog 中显示。
具体代码如下所示：

```
/* 上传文件至 Server 的方法 */
private void uploadFile()
{
  String end = "\r\n";
  String twoHyphens = "--";
  String boundary = "*****";
  try
  {
    URL url =new URL(actionUrl);
    HttpURLConnection con=(HttpURLConnection)url.openConnection();
    /* 允许 Input、Output，不使用 Cache */
    con.setDoInput(true);
    con.setDoOutput(true);
    con.setUseCaches(false);
    /* 设置传送的 method=POST */
    con.setRequestMethod("POST");
    /* setRequestProperty */
    con.setRequestProperty("Connection", "Keep-Alive");
    con.setRequestProperty("Charset", "UTF-8");
    con.setRequestProperty("Content-Type",
                    "multipart/form-data;boundary="+boundary);
    /* 设置 DataOutputStream */
    DataOutputStream ds =
      new DataOutputStream(con.getOutputStream());
    ds.writeBytes(twoHyphens + boundary + end);
    ds.writeBytes("Content-Disposition: form-data; " +
               "name=\"file1\";filename=\"" +
               newName +"\"" + end);
    ds.writeBytes(end);
    /* 取得文件的 FileInputStream */
    FileInputStream fStream = new FileInputStream(uploadFile);
    /* 设置每次写入 1024byte */
    int bufferSize = 1024;
    byte[] buffer = new byte[bufferSize];
    int length = -1;
    /* 从文件读取数据至缓冲区 */
    while((length = fStream.read(buffer)) != -1)
    {
      /* 将资料写入 DataOutputStream 中 */
      ds.write(buffer, 0, length);
    }
    ds.writeBytes(end);
    ds.writeBytes(twoHyphens + boundary + twoHyphens + end);

    /* close streams */
```

```
        fStream.close();
        ds.flush();
        /* 取得 Response 内容 */
        InputStream is = con.getInputStream();
        int ch;
        StringBuffer b =new StringBuffer();
        while( ( ch = is.read() ) != -1 )
        {
          b.append( (char)ch );
        }
        /* 将 Response 显示于 Dialog */
        showDialog(b.toString().trim());
        /* 关闭 DataOutputStream */
        ds.close();
      }
      catch(Exception e)
      {
        showDialog(""+e);
      }
    }
```

(3) 定义方法 showDialog(String mess)，用于显示提示对话框。具体代码如下所示：

```
    /* 显示 Dialog 的 method */
    private void showDialog(String mess)
    {
      new AlertDialog.Builder(example9.this).setTitle("Message")
       .setMessage(mess)
       .setNegativeButton("确定",new DialogInterface.OnClickListener()
       {
         public void onClick(DialogInterface dialog, int which)
         {
         }
       })
       .show();
    }
  }
```

至此整个展示结束，执行后的效果如图 13-11 所示，单击“开始上传”按钮后，能够将指定文件上传到服务器。



图 13-11　执行效果

13.9 远程下载安装 Android 程序

题　目	目　的	源码路径
题目 8	在 Android 中远程下载并安装 Android 程序	“光盘:\daima\13\example11”文件夹

13.9.1 我的想法

APK 是 Android Package 的缩写，即 Android 安装包。APK 是类似 Symbian Sis 或 Sisx 的文件格式，通过将 APK 文件直接传到 Android 模拟器或 Android 手机中执行安装程序即可完成安装。

我希望实例运行后，能够远程下载指定网址的 Android 应用程序，下载到手机后打开 application installer 来安装这个软件。在具体实现上，先设置一个 EditText 来获取远程程序的 URL，然后通过自定义按钮打开下载程序(使用 java.net 的 URLConnection 对象来创建连接，通过 InputStream 将下载文件写入到存储卡的缓存)。下载后通过自定义方法 openFile()打开文件，并根据文件扩展名判断是否为 APK 格式，如果是，则启动内置的 Install 程序，开始进行安装。安装完成后，在离开 Install 时通过方法 delFile()将存储卡中的临时文件删除。

13.9.2 具体实现

编写的主程序文件是 example11.java，其实现流程如下。

(1) 定义 setOnClickListener，用于监听按钮单击事件，设置文件下载到 local 端取得要安装程序的文件名称。具体代码如下所示：

```
      public void onClick(View v)
      {
        /* 文件会下载至 local 端 */
        mTextView01.setText("下载中...");
        strURL = mEditText01.getText().toString();
        /*取得欲安装程序之文件名称*/
        fileEx = strURL.substring(strURL.lastIndexOf(".")
        +1,strURL.length()).toLowerCase();
        fileNa = strURL.substring(strURL.lastIndexOf("/")
        +1,strURL.lastIndexOf("."));
        getFile(strURL);
       }
     }
    );
```

(2) 如果框中的远程地址为空则输出“请输入 URL”的提示。具体代码如下所示：

```
    mEditText01.setOnClickListener(new EditText.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
```

```
        // TODO Auto-generated method stub
        mEditText01.setText("");
        mTextView01.setText("远程安装程序(请输入 URL)");
      }
    });
  }
```

(3) 定义方法 getFile(final String strPath)，用于获取下载的 URL 文件，有异常则输出提示。具体代码如下所示：

```
/* 处理下载 URL 文件自定义函数 */
private void getFile(final String strPath)  {
  try
  {
    if (strPath.equals(currentFilePath) )
    {
      getDataSource(strPath);
    }
    currentFilePath = strPath;
    Runnable r = new Runnable()
    {
      public void run()
      {
        try
        {
          getDataSource(strPath);
        }
        catch (Exception e)
        {
          Log.e(TAG, e.getMessage(), e);
        }
      }
    };
    new Thread(r).start();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}
```

(4) 定义方法 getDataSource(String strPath)，用于获取远程文件，具体流程如下。

第 1 步：如果 URL 错误则输出提示并通过 myURL 取得 URL。

第 2 步：创建连接对象 conn，通过 File 创建临时文件。

第 3 步：取得暂存路径并将文件写入暂存 SD 存储卡。

第 4 步：通过 openFile(myTempFile)打开文件进行安装。

具体代码如下所示：

```
 /*取得远程文件*/
private void getDataSource(String strPath) throws Exception
{
  if (!URLUtil.isNetworkUrl(strPath))
```

```
  {
    mTextView01.setText("错误的 URL");
  }
  else
  {
    /*取得 URL*/
    URL myURL = new URL(strPath);
    /*创建连接*/
    URLConnection conn = myURL.openConnection();
    conn.connect();
    /*InputStream 下载文件*/
    InputStream is = conn.getInputStream();
    if (is == null)
    {
      throw new RuntimeException("stream is null");
    }
    /*创建临时文件*/
    File myTempFile = File.createTempFile(fileNa, "."+fileEx);
    /*取得暂存盘的路径*/
    currentTempFilePath = myTempFile.getAbsolutePath();
    /*将文件写入暂存盘*/
    FileOutputStream fos = new FileOutputStream(myTempFile);
    byte buf[] = new byte[128];
    do
    {
      int numread = is.read(buf);
      if (numread <= 0)
      {
        break;
      }
      fos.write(buf, 0, numread);
    }while (true);

    /*打开文件进行安装*/
    openFile(myTempFile);
    try
    {
      is.close();
    }
    catch (Exception ex)
    {
      Log.e(TAG, "error: " + ex.getMessage(), ex);
    }
  }
}
```

(5) 定义方法 openFile(File f)，设置在手机上打开文件，具体流程如下。

第 1 步：调用 getMimeType()来取得 MimeType，设置 intent 的 file 与 MimeType。

第 2 步：判断文件 MimeType 的 method 并取得扩展名。

第 3 步：根据扩展名的类型决定 MimeType。

第 4 步：如果无法直接打开则弹出软件列表给用户进行选择。

具体代码如下所示：

```
/* 在手机上打开文件的 method */
private void openFile(File f)
{
  Intent intent = new Intent();
  intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
  intent.setAction(android.content.Intent.ACTION_VIEW);

  /* 调用 getMIMEType()来取得 MimeType */
  String type = getMIMEType(f);
  /* 设置 intent 的 file 与 MimeType */
  intent.setDataAndType(Uri.fromFile(f),type);
  startActivity(intent);
}
/* 判断文件 MimeType 的 method */
private String getMIMEType(File f)
{
  String type="";
  String fName=f.getName();
  /* 取得扩展名 */
  String end=fName.substring(fName.lastIndexOf(".")
  +1,fName.length()).toLowerCase();
  /* 依扩展名的类型决定 MimeType */
  if(end.equals("m4a")||end.equals("mp3")||end.equals("mid")||
  end.equals("xmf")||end.equals("ogg")||end.equals("wav"))
  {
    type = "audio";
  }
  else if(end.equals("3gp")||end.equals("mp4"))
  {
    type = "video";
  }
  else if(end.equals("jpg")||end.equals("gif")||end.equals("png")||
  end.equals("jpeg")||end.equals("bmp"))
  {
    type = "image";
  }
  else if(end.equals("apk"))
  {
    /* android.permission.INSTALL_PACKAGES */
    type = "application/vnd.android.package-archive";
  }
  else
  {
    type="*";
  }
  /*如果无法直接打开，就弹出软件列表给用户选择 */
  if(end.equals("apk"))
  {
  }
  else
  {
    type += "/*";
  }
  return type;
}
```

(6) 定义方法 delFile(String strFileName)，用于删除 SD 卡上的临时文件。具体代码如下：

```
/*自定义删除文件方法*/
private void delFile(String strFileName)
{
  File myFile = new File(strFileName);
  if(myFile.exists())
  {
    myFile.delete();
  }
}
```

(7) 定义方法 onPause()和 onResume()，分别实现 onPause 和 onResume 状态。具体代码如下所示：

```
/*当 Activity 处于 onPause 状态时，更改 TextView 文字状态*/
@Override
protected void onPause()
{
  mTextView01 = (TextView)findViewById(R.id.myTextView1);
  mTextView01.setText("下载成功");
  super.onPause();
}

/*当 Activity 处于 onResume 状态时,删除临时文件*/
@Override
protected void onResume()
{
  // TODO Auto-generated method stub
  /* 删除临时文件 */
  delFile(currentTempFilePath);
  super.onResume();
}
}
```

至此整个展示结束，执行后将在文本框中显示目标安装程序的路径如图 13-12 所示。实例中的默认路径是 http://mz.ruan8.com/soft/2/sougoushoujishurufa_7786.apk，这是一个搜狗输入法程序。单击“按下开始安装”按钮后开始下载目标文件，如图 13-13 所示。



图 13-12　下载目标文件

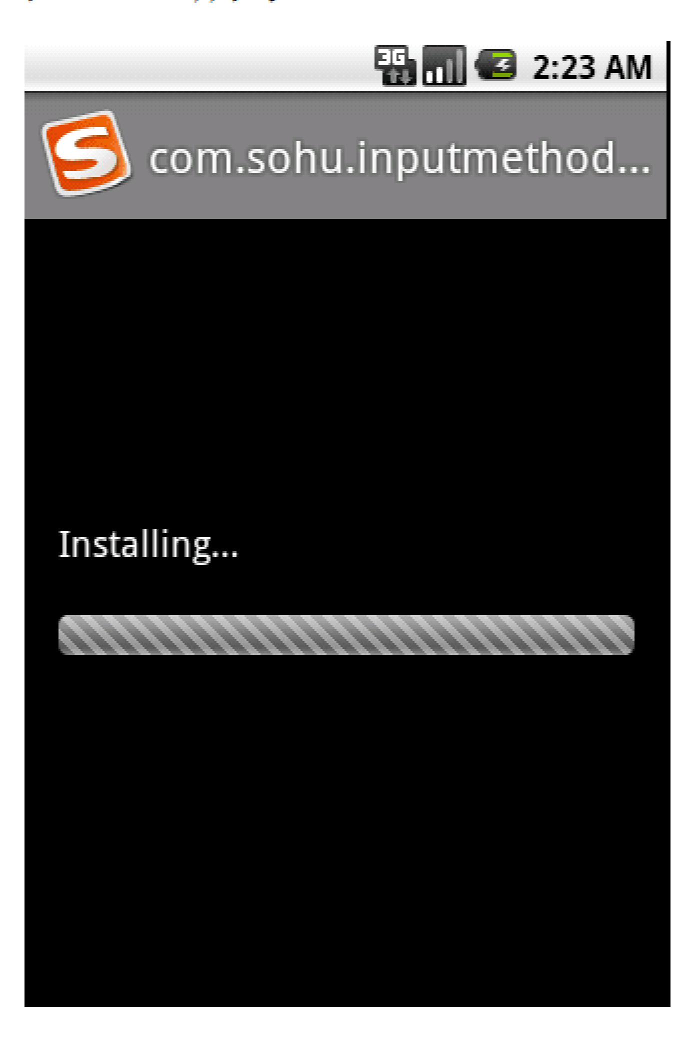


图 13-13　下载界面

下载完成后弹出安装界面，如图 13-14 所示。单击图 13-14 中的 Install 按钮后开始进行安装，安装完成后输出提示，安装完成界面如图 13-15 所示。

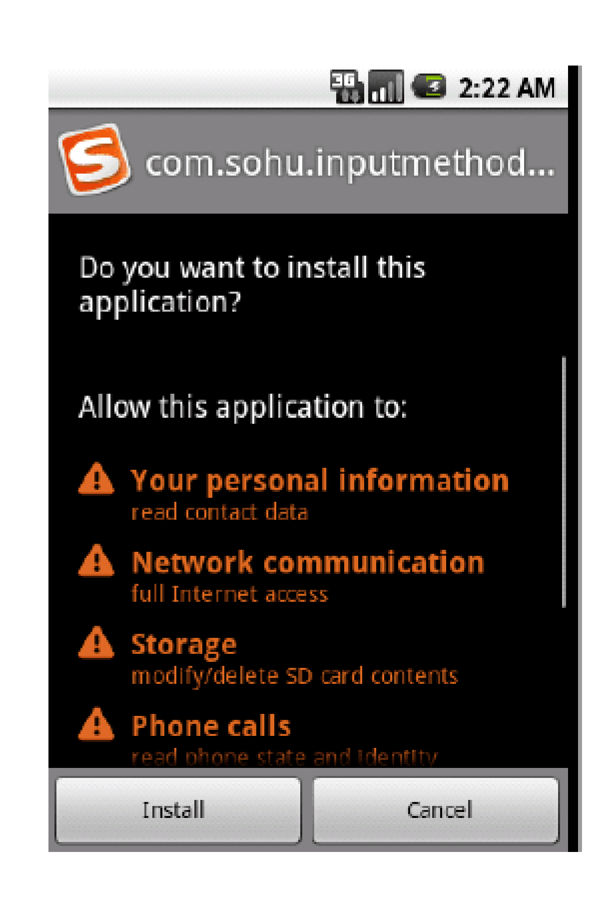


图 13-14 安装界面



图 13-15 安装完成界面

13.10 下载观看 3gp 视频

题　目	目　的	源码路径
题目 9	在 Android 中远程下载观看 3gp 视频	“光盘:\daima\13\example12”文件夹

13.10.1 我的想法

远程观看在线视频是当前智能手机的主要功能之一。因为视频文件一般比较大，所以首先得保证手机有足够的存储空间，还要确保下载的视频能被 MediaPlayer 软件所支持。我决定使用 EditText 来获取远程视频的 URL，将网址的视频下载到手机的存储卡中，以暂存的方式来进行保存，然后通过控制按钮来控制对视频的操作。在播放完毕并终止程序后，将暂存到 SD 卡中的临时视频删除。

13.10.2 具体实现

编写的主程序文件是 example12.java，其实现流程如下。

(1) 识别 MediaPlayer 是否已被释放，MediaPlayer 是否正处于暂停，用 LogCat 输出 TAG filter。具体代码如下所示：

```
/* 识别 MediaPlayer 是否已被释放*/
private boolean bIsReleased = false;

/* 识别 MediaPlayer 是否正处于暂停*/
private boolean bIsPaused = false;

/* LogCat 输出 TAG filter */
private static final String TAG = "HippoMediaPlayer";
```

```
private String currentFilePath = "";
private String currentTempFilePath = "";
private String strVideoURL = "";
```

(2)　设置播放视频的 URL 地址，用 mSurfaceView01 绑定 Layout 上的 SurfaceView，然后设置 PixnelFormat，并设置 SurfaceHolder 为 Layout SurfaceView。具体代码如下所示：

```
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  /* 将.3gp 图像文件存放 URL 网址*/
  strVideoURL =
  "http://new4.sz.3gp2.com//20100205xyy/喜羊羊与灰太狼%20 踩高跷(www.3gp2.com).3gp";
  //http://www.dubblogs.cc:8751/Android/Test/Media/3gp/test2.3gp

  mTextView01 = (TextView)findViewById(R.id.myTextView1);
  mEditText01 = (EditText)findViewById(R.id.myEditText1);
  mEditText01.setText(strVideoURL);

  /* 绑定 Layout 上的 SurfaceView */
  mSurfaceView01 = (SurfaceView) findViewById(R.id.mSurfaceView1);

  /* 设置 PixnelFormat */
  getWindow().setFormat(PixelFormat.TRANSPARENT);

  /* 设置 SurfaceHolder 为 Layout SurfaceView */
  mSurfaceHolder01 = mSurfaceView01.getHolder();
  mSurfaceHolder01.addCallback(this);
```

(3)　为影片设置大小比例，设置四个控制按钮：mPlay、mReset、mPause 和 mStop。具体代码如下所示：

```
  /* 由于原有的影片 Size 较小，故指定其为固定比例*/
  mSurfaceHolder01.setFixedSize(160, 128);
  mSurfaceHolder01.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
  mPlay = (ImageButton) findViewById(R.id.play);
  mReset = (ImageButton) findViewById(R.id.reset);
  mPause = (ImageButton) findViewById(R.id.pause);
  mStop = (ImageButton) findViewById(R.id.stop);
```

(4)　定义 mPlay.setOnClickListener，用于播放监听处理。具体代码如下所示：

```
  /* 播放按钮*/
  mPlay.setOnClickListener(new ImageButton.OnClickListener()
  {
    public void onClick(View view)
    {
      if(checkSDCard())
      {
        strVideoURL = mEditText01.getText().toString();
        playVideo(strVideoURL);
```

```
        mTextView01.setText(R.string.str play);
      }
      else
      {
        mTextView01.setText(R.string.str_err_nosd);
      }
    }
  });
```

(5) 定义 mReset.setOnClickListener，用于重新播放监听处理。具体代码如下所示：

```
  /* 重新播放按钮*/
  mReset.setOnClickListener(new ImageButton.OnClickListener()
  {
    public void onClick(View view)
    {
      if(checkSDCard())
      {
        if(bIsReleased == false)
        {
          if (mMediaPlayer01 != null)
          {
            mMediaPlayer01.seekTo(0);
            mTextView01.setText(R.string.str_play);
          }
        }
      }
      else
      {
        mTextView01.setText(R.string.str_err_nosd);
      }
    }
  });
```

(6) 定义 mPause.setOnClickListener，用于暂停播放监听处理。具体代码如下所示：

```
  /* 暂停按钮*/
  mPause.setOnClickListener(new ImageButton.OnClickListener()
  {
    public void onClick(View view)
    {
      if(checkSDCard())
      {
        if (mMediaPlayer01 != null)
        {
          if(bIsReleased == false)
          {
            if(bIsPaused==false)
            {
              mMediaPlayer01.pause();
              bIsPaused = true;
              mTextView01.setText(R.string.str_pause);
            }
```

```
          else if(bIsPaused==true)
          {
            mMediaPlayer01.start();
            bIsPaused = false;
            mTextView01.setText(R.string.str_play);
          }
        }
      }
    }
    else
    {
      mTextView01.setText(R.string.str err nosd);
    }
  }
});
```

(7) 定义 mStop.setOnClickListener，用于停止播放监听处理。具体代码如下所示：

```
  /* 终止按钮*/
  mStop.setOnClickListener(new ImageButton.OnClickListener()
  {
    public void onClick(View view)
    {
      if(checkSDCard())
      {
        try
        {
          if (mMediaPlayer01 != null)
          {
            if(bIsReleased==false)
            {
              mMediaPlayer01.seekTo(0);
              mMediaPlayer01.pause();
              mTextView01.setText(R.string.str stop);
            }
          }
        }
        catch(Exception e)
        {
          mTextView01.setText(e.toString());
          Log.e(TAG, e.toString());
          e.printStackTrace();
        }
      }
      else
      {
        mTextView01.setText(R.string.str err nosd);
      }
    }
  });
}
```

(8) 定义方法 playVideo()，用于下载指定 URL 影片并实现播放处理。具体流程如下。

第 1 步：判断传入的 strPath 是否是现有播放的连接。

第 2 步：重新构建 MediaPlayer 对象并设置播放音量。

具体代码如下所示：

```
/* 自定义下载 URL 影片并播放*/
private void playVideo(final String strPath)
{
  try
  {
    /* 若传入的 strPath 为现有播放的连接，则直接播放*/
    if (strPath.equals(currentFilePath) && mMediaPlayer01 != null)
    {
      mMediaPlayer01.start();
      return;
    }
    else if(mMediaPlayer01 != null)
    {
      mMediaPlayer01.stop();
    }

    currentFilePath = strPath;
    /* 重新构建 MediaPlayer 对象*/
    mMediaPlayer01 = new MediaPlayer();
    /* 设置播放音量*/
    mMediaPlayer01.setAudioStreamType(2);

    /* 设置显示于 SurfaceHolder */
    mMediaPlayer01.setDisplay(mSurfaceHolder01);

    mMediaPlayer01.setOnErrorListener
    (new MediaPlayer.OnErrorListener()
    {
      @Override
      public boolean onError(MediaPlayer mp, int what, int extra)
      {
        // TODO Auto-generated method stub
        Log.i
        (
          TAG,
          "Error on Listener, what: " + what + "extra: " + extra
        );
        return false;
      }
    });
```

(9) 定义 onBufferingUpdate，用于监听缓冲进度。具体代码如下所示：

```
    mMediaPlayer01.setOnBufferingUpdateListener
    (new MediaPlayer.OnBufferingUpdateListener()
    {
      @Override
      public void onBufferingUpdate(MediaPlayer mp, int percent)
```

```
    {
      // TODO Auto-generated method stub
      Log.i
      (
        TAG, "Update buffer: " +
        Integer.toString(percent) + "%"
      );
    }
  });
```

(10) 分别用 setOnPreparedListener 和 setOnCompletionListener 监听播放处理。具体代码如下所示：

```
  mMediaPlayer01.setOnCompletionListener
  (new MediaPlayer.OnCompletionListener()
  {
    @Override
    public void onCompletion(MediaPlayer mp)
    {
      // TODO Auto-generated method stub
      Log.i(TAG,"mMediaPlayer01 Listener Completed");
      mTextView01.setText(R.string.str_done);
    }
  });
  mMediaPlayer01.setOnPreparedListener
  (new MediaPlayer.OnPreparedListener()
  {
    @Override
    public void onPrepared(MediaPlayer mp)
    {
      // TODO Auto-generated method stub
      Log.i(TAG,"Prepared Listener");
    }
  });
```

(11) 定义方法 run()，用于接收连接并记录线程信息。首先在线程运行中，调用自定义函数抓下文件，当下载完成后调用 prepare，当有异常时，则输出错误信息。具体代码如下所示：

```
  Runnable r = new Runnable()
  {
    public void run()
    {
      try
      {
        /* 在线程运行中，调用自定义函数抓下文件*/
        setDataSource(strPath);
        /* 下载完后才会调用 prepare */
        mMediaPlayer01.prepare();
        Log.i
        (
          TAG, "Duration: " + mMediaPlayer01.getDuration()
        );
        mMediaPlayer01.start();
```

```
          bIsReleased = false;
        }
        catch (Exception e)
        {
          Log.e(TAG, e.getMessage(), e);
        }
      }
    };
    new Thread(r).start();
  }
  catch(Exception e)
  {
    if (mMediaPlayer01 != null)
    {
      mMediaPlayer01.stop();
      mMediaPlayer01.release();
    }
  }
}
```

(12) 定义方法 setDataSource()，用于线程启动的方式播放视频。具体代码如下所示：

```
/* 自定义 setDataSource，由线程启动*/
private void setDataSource(String strPath) throws Exception
{
  if (!URLUtil.isNetworkUrl(strPath))
  {
    mMediaPlayer01.setDataSource(strPath);
  }
  else
  {
    if(bIsReleased == false)
    {
      URL myURL = new URL(strPath);
      URLConnection conn = myURL.openConnection();
      conn.connect();
      InputStream is = conn.getInputStream();
      if (is == null)
      {
        throw new RuntimeException("stream is null");
      }
      File myFileTemp = File.createTempFile
      ("hippoplayertmp", "."+getFileExtension(strPath));

      currentTempFilePath = myFileTemp.getAbsolutePath();

      /*currentTempFilePath = /sdcard/mediaplayertmp393213.dat */

      FileOutputStream fos = new FileOutputStream(myFileTemp);
      byte buf[] = new byte[128];
      do
      {
        int numread = is.read(buf);
```

```
        if (numread <= 0)
        {
          break;
        }
        fos.write(buf, 0, numread);
      }while (true);
      mMediaPlayer01.setDataSource(currentTempFilePath);
      try
      {
        is.close();
      }
      catch (Exception ex)
      {
        Log.e(TAG, "error: " + ex.getMessage(), ex);
      }
    }
  }
}
```

(13) 定义方法 getFileExtension()，用于获取视频的扩展名。具体代码如下所示：

```
private String getFileExtension(String strFileName)
{
  File myFile = new File(strFileName);
  String strFileExtension=myFile.getName();
  strFileExtension=(strFileExtension.substring
  (strFileExtension.lastIndexOf(".")+1)).toLowerCase();

  if(strFileExtension=="")
  {
    /* 若无法顺利取得扩展名，默认为.dat */
    strFileExtension = "dat";
  }
  return strFileExtension;
}
```

(14) 定义方法 checkSDCard()，用于判断存储卡是否存在。具体代码如下所示：

```
private boolean checkSDCard()
{
  /* 判断存储卡是否存在*/
  if(android.os.Environment.getExternalStorageState().equals
  (android.os.Environment.MEDIA_MOUNTED))
  {
    return true;
  }
  else
  {
    return false;
  }
}

@Override
```

```
  public void surfaceChanged
  (SurfaceHolder surfaceholder, int format, int w, int h)
  {
    // TODO Auto-generated method stub
    Log.i(TAG, "Surface Changed");
  }

  @Override
  public void surfaceCreated(SurfaceHolder surfaceholder)
  {
    // TODO Auto-generated method stub
    Log.i(TAG, "Surface Changed");
  }

  @Override
  public void surfaceDestroyed(SurfaceHolder surfaceholder)
  {
    // TODO Auto-generated method stub
    Log.i(TAG, "Surface Changed");
  }
}
```

至此整个展示结束，执行后在文本框中显示指定播放视频的 URL，当下载完毕后实现播放处理，执行效果如图 13-16 所示。

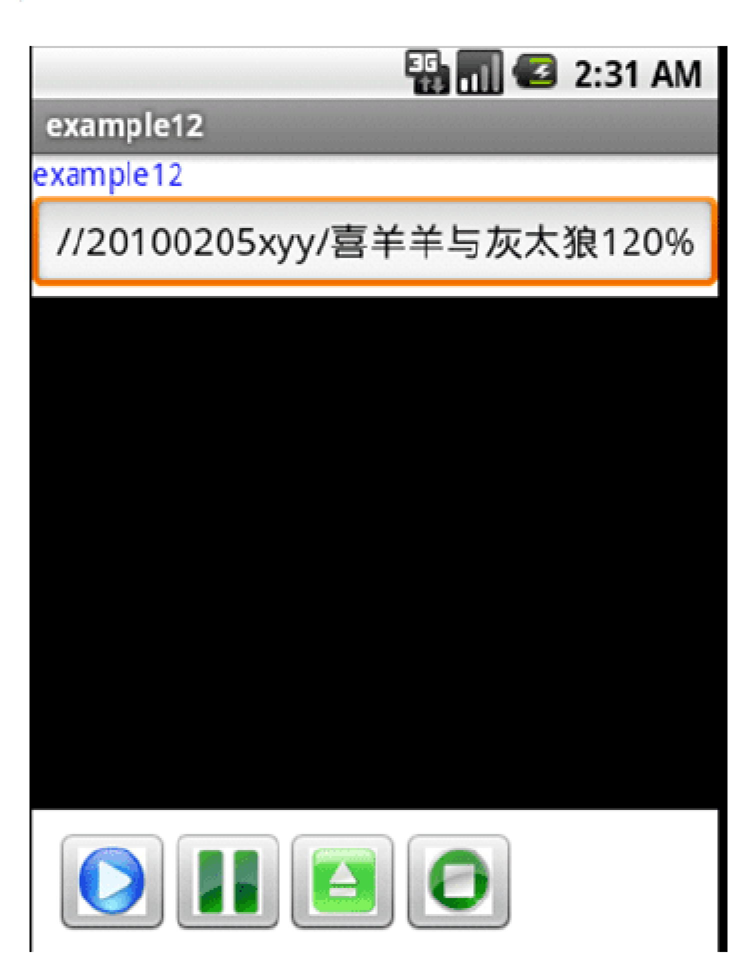


图 13-16　执行效果



第四篇　提　高　篇

第 14 章　程序也需要优化

作为任何一个程序，无论是简单还是复杂，都可能有多种编码格式，不同的编码格式肯定效率会不同。不知从何时起，程序优化提到了日常议程，但事实证明，经过优化处理后的程序，其运行速度和效率大大提高了。试想作为一个访问量巨大的项目，如果每个访问速度快那么一点点，在海量访问的前提下，速度会快很多。在本章的内容中，将简要讲解优化 Android 程序的基本知识。

14.1　9 条基础规则

以前做项目，往往看到功能后就马上开始了编码工作。结果呢？经常因为忘记编码规范，在后期耗费大量精力反复修改。犯过几次错误之后我明白了编码规范的重要性，主要有以下四个原因。

(1)　一个软件的生命周期中，80%的花费在于维护。

(2)　几乎没有任何一款软件，在其整个生命周期中，均由最初的开发人员来维护。

(3)　编码规范可以改善软件的可读性，让程序员快速并彻底地理解新的代码。

(4)　如果你将源码作为产品发布，就需要确认它是否被很好地打包并且清晰无误，一如你已构建的其他任何产品。

为了执行规范，每个软件开发人员必须一致遵守编码规范。我总结了实现 Java 高效编程的 9 条基础规则。

第 1 条：避免使用 New 关键字来创建 String 对象。把一个 String 常量 copy 到 String 对象中通常是多余的。例如下面的代码：

```
public class test{
 public void method(){
```

```
  System.out.print (str);
 }
 private String str = new String ("1");    //这里新建对象是完全没有必要的
  private String str2="2"                  //正确的应该如此
}
```

第 2 条：避免使用不必要的嵌套，过多的嵌套会使你的代码复杂化，减弱可读性。例如下面的代码：

```
Public class test {
 String add (){
  Int c=(a=a+b)+b; //过于复杂
  Return c
 }
}
```

第 3 条：避免在同一行中声明不同类型的多个变量，这样可以使程序避免混乱。例如下面的代码：

```
private int index, index1[];
```

正确的写法应该如下：

```
private int index;
private int index1[];
```

第 4 条：在每一行里写一条语句(for 语句除外)，这样可以增加代码的可读性。例如下面的代码：

```
public class OSPL {
 int method (int a, int b) {
  int i = a + b; return i; // 可读性不强
 }
```

正确的写法如下：

```
public class OSPLFixed {
 int method (int a, int b) {
  int i = a + b;
  return i;
 }
}
```

第 5 条：经常从 finalize ()中调用 super.finalize ()，此处的 finalize()是 Java 在进行垃圾收集的时候调用的，和 finally 不一样。如果你的父类没有定义 finally()的话，你也应该调用，主要有以下两个原因。

(1) 在不改变代码的情况下能够将父类的 finally 方法加到你的类中。

(2) 养成习惯调用父类的 finally 方法，即使父类没有定义 finally 方法的时候。

正确的方法如下面的代码：

```
public class parentFinalize {
 protected void finalize () throws Throwable {
  super.finalize(); // FIXED
 }
```

第 6 条：不要在 finalize ()中注销 listeners。finalize ()只有在没有对象引用的时候调用，如果 listeners 从 finalize()方法中去除了，被 finalize 的对象将不会在垃圾收集中去除。例如下面的代码：

```
public void finalize () throws Throwable {
  bButton.removeActionListener (act);
}
```

第 7 条：不要显式调用 finalize ()方法。虽然显式调用这个方法可以确保你的调用，但是当这个方法收集了以后，垃圾收集会再收集一次。例如下面的代码：

```
public class T7 {
 public void finalize() throws Throwable {
  close_resources ();
  super.finalize ();
 }
 public void close_resources() {}
}
class Test {
 void cleanup () throws Throwable {
  t71.finalize(); // 调用
  t71 = null;
 }
  private t71 = new T7 ();
}
```

对于这样的调用我们应该自己创建一个释放的方法做最初 finalize ()所做的事情，当你每次想显式调用 finalize ()的时候实际上调用了释放方法。然后再使用一个判断字段来确保这个方法只执行一次，以后再调用就没关系了。例如下面的代码：

```
public class T7 {
 public synchronized void release () throws Throwable{
  if (! released) {
   close_resources (); // do what the old 'finalize ()'
   did _released = true;
  }
 }
  public void finalize () throws Throwable {
  release ();
   super.finalize ();
 }
 public void close_resources() {}
  private boolean _released = false;
}
class TestFixed {
 void closeTest () throws Throwable {
  t71 .release (); // FIXED
  t71 = null;
 }
 private T7 t71 = new T7 ();
}
```

第 8 条：不要使用不推荐的 API，尽量使用 JDK 推荐的 API。在类和方法或者 Java 组件里有很多方法是陈旧的或者是可以选择的。有一些方法 SUN 用了 deprecated 标记，建议读者最好不要使用。例如下面的代码：

```
private List t_list = new List ();
t_list.addItem (str);
```

如果查一下 javadoc，会发现建议用 add()来代替 addItem()。

第 9 条：为所有序列化的类创建一个 serialVersionUID，可以避免你从各种不同的类中破坏序列的兼容性。如果你不特别制订一个 UID 的话，那么系统会自动产生一个 UID(根据类的内容)。如果 UID 在你新版本的类中改变了，即使那个被序列化的类没改变，你也不能反序列化老版本了。例如下面的代码：

```
public class DUID implements java.io.Serializable { public void method () {}}
```

在里面加一个 UID，当这个类的序列化形式改变的时候，你也改变这个 UID 就可以了。例如下面的代码：

```
public class DUIDFixed implements java.io.Serializable {
 public void method () {}
 private static final long serialVersionUID = 1;
}
```

14.2　必知必会命名规范

曾经认为命名可以根据个人喜好随便起，现在才知道命名也要遵循一定的规范，这是程序开发所必须遵循的根本。遵循科学的命名规范，可以提高程序的运行效率，便于程序的后期维护。师傅传授给我 6 条基本的命名规范，具体如下。

(1)　对包的命名规范

包名的前缀应该全部是小写英文字母组成，例如 java.io。

(2)　对类的命名规范

类名应使用单词，首字母需大写，若由多个单词组成，每个单词的首字母大写，尽量使类名简洁而富于描述，例如 RandomAccessFile。

(3)　对接口的命名规范

与类的命名规范相同，例如 FileFilter。当在任何需要进行命名的情况下，建议不要使用汉字或其他语言中的文字来命名，虽然在 Java 中是允许使用的。

(4)　对常量的命名规范

常量名应使用大写字母，单词间用下划线隔开，并接后见其名能知其意，例如 MAX_VALUE，常量用来储存一个最大值。

(5)　对变量的命名规范

变量名应小写且要有意义，尽量避免使用单个字符，否则遇到该变量时很难理解其用途，对于临时的变量，例如，记忆循环语句中的循环次数，通常可命名为 i、j、k 这样的单字符变量名。变量名应简短且富于描述以便容易记忆，例如用 age 变量来储存年龄。

(6) 对方法的命名规范

方法被调用来执行一个操作，因此方法名应是对该操作的描述。方法名的首字母应该小写，如果由多个单词组成，则其后单词的首字母大写。例如，一个向数据库添加数据的方法可命名为 addData()。

14.3 优 秀 代 码

都说“宝剑锋从磨砺出，梅花香自苦寒来”，但是现实是急功近利，仅仅几行代码、几个功能调试成功后，就开始大声宣布已经掌握开发技术了。其实掌握的只是皮毛，浩瀚武功博大精深，只有循序渐进加融会贯通，才能掌握真谛。师傅总结道：编写出科学、合理、高效的代码，才是成为一个优秀程序员的标志。

编写优秀代码的技巧如下。

1. 使用好的编码风格和合理的设计

在投入到编码工作中之前，先考虑大体的设计方案，这也非常关键的。“最好的计算机程序的文本是结构清晰的。”从实现一套清晰的 API、一个逻辑系统结构以及一些定义良好的组件角色与责任开始入手，将使你避免以后处理头疼的局面。我们可以通过采用良好的编程风格，来防范大多数编码错误。例如选用有意义的变量名，或者审慎地使用括号，都可以使编码变得更加清晰明了，并减少缺陷出现的可能性。

2. 不要仓促地编写代码

急功近利式的编程习惯很多人都有，使用这种编程方式的程序员会很快地开发出一个函数，马上把这个函数交给编译器来检查语法，接着运行一遍看看能不能用，然后就进入下一个任务，这种方式充满了危险。相反，应该在写每一行代码时都三思而后行。想一想可能会出现什么样的错误，你是否已经考虑了所有可能出现的逻辑分支？放慢速度，有条不紊的编程虽然看上去很平凡，但这的确是减少代码缺陷的好办法。

因此，一定要在完成与一个代码段相关的所有任务之后，再进入下一个环节。例如，如果你决定先编写主体部分，再加入错误检查和处理，那么一定要确保这两项工作的完成都遵循章法。如果你要推迟错误检查的编写，而直接开始编写超过三个代码段的主体部分，你一定要慎之又慎。你也许真的想随后再回来编写错误检查，但却一而再再而三地向后推迟，这期间你可能会忘记很多上下文，使得接下来的工作更加耗时和琐碎。

3. 编译时打开所有警告开关

大多数语言的编译器都会在受伤到一定程度后给出一大堆错误信息。建议在任何情况下都要打开编译器的警告功能，如果你的代码产生了任何的警告信息，立即修正代码，让编译器的报错声停下来。在启用了警告功能之后，不要对不能安静地完成编译的代码感到满意。警告的出现总是有原因的。即使你认为某个警告无关紧要，也不要置之不理，否则，总有一天这个警告会隐藏一个确实重要的警告。

4. 编码要清晰

如果要你从简洁(但是有可能让人困惑)的代码和清晰(但是有可能比较冗长)的代码中进行选择，一定要选那些看上去和预期相符合的代码，即使它不太优雅。例如，将复杂的代数运算拆分为一系列单独的语句，使逻辑更清晰。简单就是一种美，不要让你的代码过于复杂。

5. 使用安全的数据结构

最常见的安全隐患大概是由缓冲溢出引起的。缓冲溢出是由于不正确地使用固定大小的数据结构而造成的。如果你的代码在没有检查一个缓冲的大小之前就写入这个缓冲，那么写入的内容总是有可能会超过缓冲的末尾的。避免由于这些隐患而受到攻击的方法就是不要编写这样的糟糕代码，使用更安全的、不允许破坏程序的数据结构，也可以对不安全的数据类型系统地使用安全的操作。

6. 使用静态分析工具

许多独立的静态分析工具可供使用，如用于 C 语言的 lint(以及更多新出的衍生工具)和用于.NET 汇编程序的 FxCop。你的日常编程工作，应该包括使用这些工具来帮助检查你的代码，它们会比你的编译器能挑出更多的错误。

7. 检查所有的返回值

如果一个函数返回一个值，它这样做肯定是有理由的。检查这个返回值，如果返回值是一个错误代码，你就必须辨别这个代码并处理所有的错误。不要让错误悄无声息地侵入你的程序，忍受错误会导致不可预知的行为，这既适用于用户自定义的函数，也适用于标准库函数。你会发现，大多数难以察觉的错误都是因为程序员没有检查返回值而出现的。不要忘记，某些函数会通过不同的机制(例如，标准 C 库的 erron)返回错误。不论何时，都要在适当的级别上捕获和处理相应的异常。

8. 谨慎处理内存

在任何编程应用中，必须彻底释放在执行期间所获取的任何资源。其中内存资源是这类资源最常提到的一个例子，但并不是唯一的一个。在编程过程中必须管理好内存，不要因为觉得操作系统会在你的程序退出时清除程序，就不注意关闭文件或释放内存。对于代码还会执行多长时间，是否会耗尽所有的文件句柄或占用所有的内存，对于程序员来说是一无所知的，甚至不能肯定操作系统是否会完全释放你的资源。

Java 和.NET 使用垃圾回收器来执行释放工作，所以程序员可以无须专门释放资源。但是，不要因此而对安全性抱有错误的想法。正确的做法是，在程序中必须显式地终止对那些不再需要，或不会被自动清除的对象的引用，不要意外地保留对对象的引用。不太先进的垃圾回收器也很容易会被循环引用蒙蔽(例如，A 引用 B，B 又引用 A，除此之外没有对 A 和 B 的引用)。这会导致对象永远不会被清除；这是一种难以发现的内存泄漏形式。

9. 使用标准语言工具

明确地定义你正在使用的是哪个语言版本。除非你的项目要求你(最好是有一个好的理由)，否则不要将命运交给编译器，或者对该语言的任何非标准的扩展。如果该语言的某个领域还没

有定义，就不要依赖你所使用的特定编译器的行为(例如，不要依赖你的 C 编译器将 char 作为有符号的值对待，因为其他的编译器并不是这样的)，这样做会产生非常脆弱的代码。当你更新了编译器之后，会发生什么？一位新的程序员加入到开发团队中，如果他不理解那些扩展，会发生什么？依赖于特定编译器的个别行为，将导致以后难以发现的错误。

10. 谨慎强制转换

大多数语言都允许你将数据从一种类型强制转换为另一种类型，这种操作有时比其他操作更成功。如果试着将一个 64 位的整数转换为较小的 8 位数据类型，那么其他的 56 位会怎么样呢？你的执行环境可能会突然抛出异常，或者悄悄地使你数据的完整性降级。很多程序员并不考虑这类事情，所以他们的程序就会表现出不正常的行为。

如果你真的想使用强制转换，就必须对之深思熟虑。你所告诉编译器的是：“忘记类型检查吧，我知道这个变量是什么，而你并不知道。”你在类型系统中撕开了一个大洞，并直接穿越过去，这样做很不可靠。如果你犯了任何一种错误，编译器将只会静静地坐在那里小声嘀咕道：“我告诉过你的。”如果你很幸运，运行时可能会抛出异常让你了解发生了错误，但这完全依赖于你要进行的是什么转换。

11. 使用好的诊断信息日志工具

当编写新的代码时，常常会加入很多诊断信息，以确定程序的运行情况。在调试结束后是否应该删除这些诊断信息呢？保留这些信息对以后再次访问代码会带来很多方便，特别是如果在此期间可以有选择地禁用这些信息，有很多诊断信息日志系统可以帮助实现这种功能。这些系统中很多都可以使诊断信息在不需要的时候不带来任何开销，可以有选择地使它们不参加编译。

14.4　程 序 优 化

对于任何一个程序来说，可供利用的内存、CPU 和带宽都是有限的。我们使用优化的目的就是让程序尽量减少对这些资源的占有，这就得需要程序优化来实现。

14.4.1　基本优化

在 Java 程序中，性能问题的大部分原因并不在于 Java 语言，而是在于程序本身。养成好的代码编写习惯非常重要，比如正确地、巧妙地运用 java.lang.String 类和 java.util.Vector 类，它能够显著地提高程序的性能。下面就来具体地分析一下这方面的问题。

(1)　尽量指定类的 final 修饰符，因为带有 final 修饰符的类是不可派生的。在 Java 核心 API 中，有许多应用 final 的例子，例如 java.lang.String。为 String 类指定 final 防止了人们覆盖 length()方法。另外，如果指定一个类为 final，则该类所有的方法都是 final。Java 编译器会寻找机会内联(inline)所有的 final 方法(这和具体的编译器实现有关)，此举能够使性能平均提高 50%。

(2)　尽量重用对象。特别是 String 对象的使用中，出现字符串连接情况时应用 StringBuffer 代替。由于系统不仅要花时间生成对象，以后可能还需花时间对这些对象进行垃圾回收和处理。因此，生成过多的对象将会给程序的性能带来很大的影响。

(3) 尽量使用局部变量，调用方法时传递的参数以及在调用中创建的临时变量都保存在栈(Stack)中，速度较快。其他变量，如静态变量、实例变量等，都在堆(Heap)中创建，速度较慢。另外，依赖于具体的编译器/JVM，局部变量还可能得到进一步优化。

(4) 不要重复初始化变量，默认情况下，调用类的构造函数时，Java 会把变量初始化成确定的值。所有的对象被设置成 null，整数变量(byte、short、int、long)设置成 0，float 和 double 变量设置成 0.0，逻辑值设置成 false。当一个类从另一个类派生时，这一点尤其应该注意，因为用 new 关键词创建一个对象时，构造函数链中的所有构造函数都会被自动调用。

(5) 在 Java + Oracle 的应用系统开发中，Java 中内嵌的 SQL 语句尽量使用大写的形式，以减轻 Oracle 解析器的解析负担。

(6) Java 编程过程中，进行数据库连接、I/O 流操作时务必小心，在使用完毕后，及时关闭以释放资源。因为对这些大对象的操作会造成系统大的开销，稍有不慎，会导致严重的后果。

(7) 由于 JVM 有其自身的 GC 机制，不需要程序开发者的过多考虑，从一定程度上减轻了开发者负担，但同时也遗漏了隐患，过分地创建对象会消耗系统的大量内存，严重时会导致内存泄露，因此，保证过期对象的及时回收具有重要意义。JVM 回收垃圾的条件是：对象不再被引用；然而，JVM 的 GC 并非十分的机智，即使对象满足了垃圾回收的条件也不一定会被立即回收，所以，建议我们在对象使用完毕后应手动置成 null。

(8) 在使用同步机制时，应尽量使用方法同步代替代码块同步。

(9) 尽量减少对变量的重复计算，例如：

```
for(int i = 0;i < list.size; i++) {
          …
}
```

应替换为：

```
for(int i = 0,int len = list.size();i < len; i++) {
          …
}
```

(10) 尽量采用 lazy loading 的策略，即在需要的时候才开始创建。例如：

```
   String str = "aaa";
          if(i == 1) {
             list.add(str);
}
```

应替换为：

```
          if(i == 1) {
String str = "aaa";
             list.add(str);
}
```

(11) 慎用异常。

异常对性能不利，抛出异常首先要创建一个新的对象。Throwable 接口的构造函数调用名为 fillInStackTrace()的本地(Native)方法，fillInStackTrace()方法检查堆栈，收集调用跟踪信息。只要有异常被抛出，VM 就必须调整调用堆栈，因为在处理过程中创建了一个新的对象。异常只能用于错误处理，不应该用来控制程序流程。

(12) 不要在循环中使用下面的代码：

```
Try {
} catch() {
}
```

应把其放置在最外层。

(13) 注意 StringBuffer 的使用。StringBuffer 表示可变的、可写的字符串。有如下三个构造方法：

```
StringBuffer ();                    //默认分配 16 个字符的空间
StringBuffer (int size);            //分配 size 个字符的空间
StringBuffer (String str);          //分配 16 个字符+str.length()个字符空间
```

可以通过 StringBuffer 的构造函数来设定它的初始化容量，这样可以明显地提升性能。这里提到的构造函数是 StringBuffer(int length)，length 参数表示当前的 StringBuffer 能保持的字符数量。你也可以使用 ensureCapacity(int minimumcapacity)方法在 StringBuffer 对象创建之后设置它的容量。首先我们看看 StringBuffer 的缺省行为，然后再找出一条更好地提升性能的途径。

StringBuffer 在内部维护一个字符数组，当你使用缺省的构造函数来创建 StringBuffer 对象的时候，因为没有设置初始化字符长度，StringBuffer 的容量被初始化为 16 个字符，也就是说缺省容量就是 16 个字符。当 StringBuffer 达到最大容量的时候，它会将自身容量增加到当前的 2 倍再加 2，也就是(2*旧值+2)。如果你使用缺省值，初始化之后接着往里面追加字符，在你追加到第 16 个字符的时候它会将容量增加到 34，也就是(2*16+2)，当追加到 34 个字符的时候就会将容量增加到 70，也就是(2*34+2)。无论何事只要 StringBuffer 到达它的最大容量，它就不得不创建一个新的字符数组然后重新将旧字符和新字符都拷贝一遍。所以总是给 StringBuffer 设置一个合理的初始化容量值是错不了的，这样会带来立竿见影的性能增益。

StringBuffer 初始化过程的调整作用由此可见一斑。所以，使用一个合适的容量值来初始化 StringBuffer 永远都是一个最佳的建议。

(14) 合理地使用 Java 类 java.util.Vector。

简单地说，一个 Vector 就是一个 java.lang.Object 实例的数组。Vector 与数组相似，它的元素可以通过整数形式的索引访问。但是，Vector 类型的对象在创建之后，对象的大小能够根据元素的增加或者删除而扩展、缩小。下面是向 Vector 加入元素的代码：

```
Object obj = new Object();
Vector v = new Vector(100000);
for(int I=0;
I<100000; I++) { v.add(0,obj); }
```

除非有绝对充足的理由要求每次都把新元素插入到 Vector 的前面，否则上面的代码对性能不利。在默认构造函数中，Vector 的初始存储能力是 10 个元素，如果新元素加入时存储能力不足，则以后存储能力每次加倍。Vector 类就像 StringBuffer 类一样，每次扩展存储能力时，所有的元素都要复制到新的存储空间中。下面的代码片段要比前面的例子快几个数量级：

```
Object obj = new Object();
Vector v = new Vector(100000);
for(int I=0; I<100000; I++) { v.add(obj); }
```

同样的规则也适用于 Vector 类的 remove()方法。由于 Vector 中各个元素之间不能含有“空

隙”，删除除最后一个元素之外的任意其他元素都导致被删除元素之后的元素向前移动。也就是说，从 Vector 删除最后一个元素要比删除第一个元素的“开销”低好几倍。

假设要从前面的 Vector 删除所有元素，我们可以使用下面的代码：

```
for(int I=0; I<100000; I++){
v.remove(0);
}
```

但是与下面的代码相比，前面的代码要慢几个数量级。

```
for(int I=0; I<100000; I++) {
v.remove(v.size()-1);
}
```

从 Vector 类型的对象 v 删除所有元素的最好方法如下：

```
v.removeAllElements();
```

假设 Vector 类型的对象 v 包含字符串“Hello”。考虑下面的代码，它要从这个 Vector 中删除“Hello”字符串，代码如下：

```
String s = "Hello";
int i = v.indexOf(s);
if(I != -1) v.remove(s);
```

这些代码看起来没什么错误，但它同样对性能不利。在这段代码中，indexOf()方法对 v 进行顺序搜索寻找字符串“Hello”，remove(s)方法也要进行同样的顺序搜索。改进后的代码是：

```
String s = "Hello";
int i = v.indexOf(s);
if(I != -1) v.remove(i);
```

在上述代码中直接在 remove()方法中给出待删除元素的精确索引位置，从而避免了第二次搜索。改进后更好的代码是：

```
String s = "Hello"; v.remove(s);
```

最后，再来看一个有关 Vector 类的代码片段：

```
for(int I=0; I++;I < v.length)
```

如果 v 包含 100 000 个元素，这个代码片段将调用 v.size()方法 100 000 次。虽然 size 方法是一个简单的方法，但它仍旧需要一次方法调用的开销，至少 JVM 需要为它配置以及清除堆栈环境。在这里，for 循环内部的代码不会以任何方式修改 Vector 类型对象 v 的大小，因此上面的代码最好改写成下面这种形式：

```
int size = v.size(); for(int I=0; I++;I<size)
```

虽然这是一个简单的改动，但它赢得了性能。毕竟每一个 CPU 周期都是宝贵的。

(15) 当复制大量数据时，使用 System.arraycopy()命令。

(16) 使用代码重构增强代码的可读性。例如下面的代码：

```
public class ShopCart {
      private List carts ;
      …
```

```
        public void add (Object item) {
            if(carts == null) {
                carts = new ArrayList();
}
carts.add(item);
}
public void remove(Object item) {
    if(carts.contains(item)) {
        carts.remove(item);
}
}
public List getCarts() {
    //返回只读列表
    return Collections.unmodifiableList(carts);
}

//不推荐这种方式
//this.getCarts().add(item);
}
```

(17) 不用 new 关键字创建类实例。

用 new 关键字创建类的实例时，构造函数链中的所有构造函数都会被自动调用，但如果一个对象实现了 Cloneable 接口，我们可以调用它的 clone()方法。clone()方法不会调用任何类构造函数。

在使用设计模式(Design Pattern)的场合，如果用 Factory 模式创建对象则改用 clone()方法创建新的对象实例会非常简单。下面是 Factory 模式的一段典型代码：

```
public static Credit getNewCredit() {
    return new Credit();
}
```

改进后的代码使用 clone()方法，代码如下所示：

```
private static Credit BaseCredit = new Credit();
public static Credit getNewCredit() {
    return (Credit) BaseCredit.clone();
}
```

上述思路对于数组处理同样很有用。

(18) 谨慎乘法和除法，具体的代码如下：

```
for (val = 0; val < 100000; val+=5) {
alterX = val * 8; myResult = val * 2;
}
```

用移位操作替代乘法操作可以极大地提高性能，下面是修改后的代码：

```
for (val = 0; val < 100000; val += 5) {
alterX = val << 3; myResult = val << 1;
}
```

修改后的代码不再做乘以 8 的操作，而是改用等价的左移 3 位操作，每左移 1 位相当于乘以 2。相应地，右移 1 位操作相当于除以 2。值得一提的是，虽然移位操作速度快，但可能使代

码比较难于理解，所以最好加上一些注释。

(19) 在 JSP 页面中关闭无用的会话。

一个常见的误解是以为 session 在有客户端访问时就被创建，然而事实是直到某 server 端程序调用 HttpServletRequest.getSession(true)这样的语句时才被创建，注意如果 JSP 没有显式地使用 <%@page session="false"%> 关闭 session，则 JSP 文件在编译成 Servlet 时将会自动加上这样一条语句 HttpSession session = HttpServletRequest.getSession(true)；这也是 JSP 中隐含的 session 对象的来历。由于 session 会消耗内存资源，因此，如果不打算使用 session，应该在所有的 JSP 中关闭它。

对于那些无须跟踪会话状态的页面，关闭自动创建的会话可以节省一些资源。使用如下 page 指令：

```
<%@ page session="false"%>
```

(20) JDBC 与 I/O。

如果应用程序需要访问一个规模很大的数据集，则应当考虑使用块提取方式。默认情况下，JDBC 每次提取 32 行数据。举例来说，假设我们要遍历一个 5000 行的记录集，JDBC 必须调用数据库 157 次才能提取到全部数据。如果把块大小改成 512，则调用数据库的次数将减少到 10 次。

(21) Servlet 与内存使用。

许多开发者随意地把大量信息保存到用户会话之中。一些时候，保存在会话中的对象没有及时地被垃圾回收机制回收。从性能上看，典型的症状是用户感到系统周期性地变慢，却又不能把原因归于任何一个具体的组件。如果监视 JVM 的堆空间，它的表现是内存占用不正常地大起大落。

解决这类内存问题主要有两种办法。第一种办法是，在所有作用范围为会话的 Bean 中实现 HttpSessionBindingListener 接口。这样，只要实现 valueUnbound()方法，就可以显式地释放 Bean 使用的资源。另外一种办法就是尽快地把会话作废。大多数应用服务器都有设置会话作废间隔时间的选项。另外，也可以用编程的方式调用会话的 setMaxInactiveInterval()方法，该方法用来设定在作废会话之前，Servlet 容器允许的客户请求的最大间隔时间，以秒计。

(22) 使用缓冲标记。

一些应用服务器加入了面向 JSP 的缓冲标记功能。例如，BEA 的 WebLogic Server 从 6.0 版本开始支持这个功能，Open Symphony 工程也同样支持这个功能。JSP 缓冲标记既能够缓冲页面片段，也能够缓冲整个页面。当 JSP 页面执行时，如果目标片段已经在缓冲之中，则生成该片段的代码就不用再执行。页面级缓冲捕获对指定 URL 的请求，并缓冲整个结果页面。对于购物篮、目录以及门户网站的主页来说，这个功能极其有用。对于这类应用，页面级缓冲能够保存页面执行的结果，供后继请求使用。

(23) 选择合适的引用机制。

在典型的 JSP 应用系统中，页头、页脚部分往往被抽取出来，然后根据需要引入页头、页脚。当前，在 JSP 页面中引入外部资源的方法主要有两种：include 指令和 include 动作。

include 指令：例如<%@ include file="copyright.html" %>，该指令在编译时引入指定的资源。在编译之前，带有 include 指令的页面和指定的资源被合并成一个文件。被引用的外部资源在编译时就确定，比运行时才确定资源更高效。

include 动作：例如<jsp:include page="copyright.jsp" />，该动作引入指定页面执行后生成的

结果。由于它在运行时完成，因此对输出结果的控制更加灵活。但时，只有当被引用的内容频繁地改变时，或者在对主页面的请求没有出现之前，被引用的页面无法确定时，使用 include 动作才合算。

(24) 及时清除不再需要的会话。

为了清除不再需要的会话，许多应用服务器都有默认的会话超时时间，一般为 30 分钟。当应用服务器需要保存更多会话时，如果内存容量不足，操作系统会把部分内存数据转移到磁盘，应用服务器也可能根据“最近最频繁使用”(Most Recently Used)算法把部分不活跃的会话转存到磁盘，甚至可能抛出“内存不足”异常。在大规模系统中，串行化会话的代价是很昂贵的。当会话不再需要时，应当及时调用 HttpSession.invalidate()方法清除会话。HttpSession.invalidate()方法通常可以在应用的退出页面调用。

(25) 不要将数组声明为：public static final。

不要将数组声明为 Public static final 的形式，这样会被当做常量来处理。

(26) HashMap 的遍历效率讨论。

经常遇到对 HashMap 中的 key 和 value 值对的遍历操作，遍历操作有如下两种方法：

```
Map<String, String[]> paramMap = new HashMap<String, String[]>();
//第一个循环
Set<String> appFieldDefIds = paramMap.keySet();
for (String appFieldDefId : appFieldDefIds) {
String[] values = paramMap.get(appFieldDefId);
......
}
//第二个循环
for(Entry<String, String[]> entry : paramMap.entrySet()){
String appFieldDefId = entry.getKey();
String[] values = entry.getValue();
.......
}
```

上述第一种实现的效率明显不如第二种高。

Set<String> appFieldDefIds = paramMap.keySet()；是先从 HashMap 中取得的 keySet。具体代码如下：

```
public Set<K> keySet() {
Set<K> ks = keySet;
return (ks != null ? ks : (keySet = new KeySet()));
}

private class KeySet extends AbstractSet<K> {
public Iterator<K> iterator() {
return newKeyIterator();
}
public int size() {
return size;
}
public boolean contains(Object o) {
return containsKey(o);
}
```

```
public boolean remove(Object o) {
return HashMap.this.removeEntryForKey(o) != null;
}
public void clear() {
HashMap.this.clear();
}
}
```

这其实是返回一个私有类 KeySet，它是从 AbstractSet 继承而来，实现了 Set 接口。接下来是 for/in 循环的语法，代码如下：

```
for(declaration : expression_r)
statement
```

在执行阶段被翻译成如下格式：

```
for(Iterator<E> #i = (expression_r).iterator(); #i.hashNext();){
declaration = #i.next();
statement
}
```

所以在第一个 for 语句 for (String appFieldDefId:appFieldDefIds) 中调用了 HashMap.keySet().iterator()，而这个方法调用了 newKeyIterator()。代码如下：

```
Iterator<K> newKeyIterator() {
return new KeyIterator();
}
private class KeyIterator extends HashIterator<K> {
public K next() {
return nextEntry().getKey();
}
}
```

因此，还是在 for 中调用了。

在第二个循环 for(Entry<String, String[]> entry : paraMap.entrySet())中使用的 Iterator，是如下的一个内部类：

```
private class EntryIterator extends HashIterator<Map.Entry<K,V>> {
public Map.Entry<K,V> next() {
return nextEntry();
}
}
```

此时第一个循环得到 key，第二个循环得到 HashMap 的 Entry。效率就是从循环里面体现出来的，第二个循环可以直接取得 key 和 value 值，而第一个循环还是得再利用 HashMap 的 get(Object key)来取得 value 值。HashMap 的 get(Object key)方法具体代码如下：

```
public V get(Object key) {
Object k = maskNull(key);
int hash = hash(k);
int i = indexFor(hash, table.length); //Entry[] table
Entry<K,V> e = table;
while (true) {
if (e == null)
```

```
return null;
if (e.hash == hash && eq(k, e.key))
return e.value;
        e = e.next;
}
}
```

其实，就是再次利用 Hash 值取出相应的 Entry 做比较得到的结果，所以使用第一种循环相当于两次进入 HashMap 的 Entry 中。而第二个循环取得 Entry 的值之后直接取得 key 和 value 值，效率比第一个循环高。按照 Map 的概念来看也应该是用第二个循环好一点，它本来就是 key 和 value 的值对，将 key 和 value 分开操作在这里不是个好选择。

(27) Array(数组)和 ArrayList 的使用，两者的特点如下。

- ❑ Array([])：最高效，但是其容量固定且无法动态改变。
- ❑ ArrayList：容量可动态增长；但牺牲效率。

基于效率和类型检验，应尽可能使用 Array，无法确定数组大小时才使用 ArrayList。ArrayList 是 Array 的复杂版本。

在 ArrayList 内部封装了一个 Object 类型的数组，从一般的意义来说，它和数组没有本质的差别，甚至与 ArrayList 的许多方法，如 Index、IndexOf、Contains、Sort 等都是在内部数组的基础上直接调用 Array 的对应方法。

当 ArrayList 存入对象时，抛弃类型信息，所有对象屏蔽为 Object，编译时不检查类型，但是运行时会报错。

注意： 从 jdk5 中加入了对泛型的支持，已经可以在使用 ArrayList 时进行类型检查。

由此可见，ArrayList 与数组的区别主要就是由于动态增容的效率问题了。

(28) 尽量使用 HashMap 和 ArrayList，除非必要，否则不推荐使用 HashTable 和 Vector，后者由于使用同步机制，而导致了性能的开销。

(29) StringBuffer 和 StringBuilder 的区别。

java.lang.StringBuffer 线程安全的可变字符序列。一个类似于 String 的字符串缓冲区，但不能修改。StringBuilder 与该类相比，通常应该优先使用 java.lang.StringBuilder 类，因为它支持所有相同的操作，但由于它不执行同步，所以速度更快。为了获得更好的性能，在构造 StirngBuffer 或 StirngBuilder 时应尽可能指定它的容量。当然，如果你操作的字符串长度不超过 16 个字符就不用了。相同情况下使用 StringBuilder 相比使用 StringBuffer 仅能获得 10%~15%的性能提升，但却要冒多线程不安全的风险。而在现实的模块化编程中，负责某一模块的程序员不一定能清晰地判断该模块是否会放入多线程的环境中运行。因此，除非你能确定你的系统的瓶颈是在 StringBuffer 上，并且确定你的模块不会运行在多线程模式下，否则还是用 StringBuffer 吧。

14.4.2　程序性能优化

传统手机软件是由手机开发厂商固化在手机中的。在 SUN 公司推出 J2ME 后，各大手机厂商很快就推出了支持 J2ME 的手机。支持 J2ME 的手机可运行由第三方提供的基于 J2ME 开发的软件，使手机的扩展功能得到极大增强。

在 J2ME 规范中，J2ME 定义了基于 Java 类库的 CDC(Connected Device Configuration)和 CLDC(Connected LimitedDevice Configuration)，在 CDC 和 CLDC 之上 J2ME 又定义了 Profile 层，称之为 MIDP(Mobile Information Device Profile)。MIDP 对 CDC / CLDC 进行一定程度的封装，并定义了一整套应用程序接口和用户接口。MIDP 为 J2ME 应用提供了两类 UI(User Interface)，分别称为高级用户界面和低级用户界面。高级用户界面是被适配到设备上由手机操作系统定义外观的通用图形组件；低级用户界面则允许开发者根据需要在界面上任意绘制图形，是由开发者完全控制显示内容的图形界面。由于手机游戏界面绝大多数由自定义的图形元素构成，所以不可避免地要采用低级用户界面。

低级用户界面开发具有高度的自由性，不同的游戏架构和不同的编码风格将对最终产品的性能产生极大影响。目前对于 J2ME 手机游戏开发而言，最大的问题在于 J2ME 运行平台有限的资源。如何有效地利用现有资源以提高游戏的运行性能，将成为开发者面临的首要问题。

目前被普遍采用的优化方案如下。

(1) 优化循环，通过将重复的子表达式重新组织来提高循环体的运行性能。

(2) 减少使用对象的数量来提高运行性能。

(3) 缩减网络传输数据来缩短等待时间等。

提高运行性能有以下四种性能优化的策略。

1. 采用对象池技术，提高对象的利用效率

Java 是面向对象的编程语言，创建和释放对象会占用相当大的资源，而在 Java 里不用对象在很多情况下又无法实现。本文提出一种对象池技术，将有效解决创建和释放对象带来的性能损失问题。

例如游戏中敌机的处理方式：一种解决方案是游戏在载入关卡的时候，为每架敌机创建一个对象，随着游戏的行进，按照游戏进程显示不同的敌机。这种方案中创建对象的资源开销巨大，因此严重影响手机游戏的运行性能。虽然在敌机被击毁的时候可以将对应的对象设置为 null 并由 System．gc()回收，但在下一关卡载入的时候还要重新创建相关的对象，增加了用户的等待时间。另一种方案是在游戏的进程中，根据需要动态创建敌机对象，被击毁后将对象设置为 null 并由 System．gc()回收。这种方案虽然能减少游戏载入时间，但是频繁地创建和释放对象的资源开销使游戏变得不流畅，对于射击游戏这类对实时性要求很高的游戏而言，这一点是不可接受的。

从研究数据来看，游戏性能损耗主要源于创建和释放对象，而不创建对象又无法实现逻辑功能，因此要尽量避免对象的创建和释放。问题的重点就转到怎样有效利用已有的对象上。本文提出的对象池技术，就是根据需求先创建一定量的对象，在需要创建对象的时候从池中申请空闲对象，释放对象时把对象释放回池中，以有效避免由创建和释放对象带来的性能损失。

分析游戏需求发现同时显示的敌机数量最多不过 5 架，采用对象池技术可以先定义一个对象池，容量为同时显示的敌机最大数量，代码如下：

```
Enemy[5]enemy=new Enemy[5];
for(int i=0; i<5: i++){
enemy[i]=new Enemy();
}
```

在类 Enemy 里增加标志属性 used 和带参数的 reset 方法使对象可重置到初始状态，在载入

游戏关卡的时候初始化对象池，在需要创建对象的时候从对象池获取一个未被使用的对象并使用 reset 方法初始化，需要释放对象的时候只需将标志位修改以供下次使用。与第一种解决方案相比使用对象池减少了相当一部分的资源开销，与第二种解决方案相比使用对象池避免了频繁创建对象的额外资源开销。

2. 尽可能地使用基本数据类型代替对象

对象虽然屏蔽了细节实现，但是是以牺牲存储空间来实现的。使用基本数据类型则仅需要少量的存储空间。虽然对象在逻辑的实现上具有优势，但是在绝大多数情况下，使用基本数据类型比使用对象更高效，所以在局部不影响逻辑的情况下可以考虑用基本数据类型代替对象实现。

在游戏中飞机要发射子弹，基于面向对象的设计模式可将子弹抽象成 Bullet 类，然后定义代表子弹的对象池，在发射子弹的时候，在对象池中查找可用对象并重置其位置为飞机所在的位置，即在每一个 gameloop 中将 Bullet 对象的纵坐标值减少一定数值(屏幕坐标系 Y 轴坐标为自上而下递增)。由于对象操作的效率低于对基本数据类型的操作，由此可以想到由基本数据类型来代替对象实现。代码如下：

```
 int[][] bullet=new int[2][10]; //假设同屏同时显示 10 颗子弹
 for(int i=0; i<10; i++){
 bullet[0][i]=999; //对应 Bullet 对象的 xposition 属性
 bullet[1][i]=999; //对应 Bullet 对象的 yposition 属性
 }
```

在每一个 gameloop 中，代码如下：

```
while(run){
 for(int i=0; i<10: i++){
 if(bullet[1][i]==999){
 //，重置为发射子弹飞机当前的位置
 bullet[0][i]==myPlane. getXposition();
 bullet[1][i]==myPlane. getYposition();
 }else{
 //，当前使用的子弹让其纵坐标减少
 bullet[1][j]-=10:
 }
 }
 }
```

3. 用简单的数值计算代替复杂的函数计算

在任何一款游戏里，都不可避免地要进行函数运算，而大量复杂的函数计算势必会占用大量的空间和处理器时间，进而影响游戏性能。减少函数计算复杂度或者减少复杂函数的调用次数甚至避免使用复杂函数计算，将有利于游戏性能的提高。与复杂函数运算相比，简单的数值计算无论从时间上还是空间上都有极大的优势。

在游戏中敌机也会发射子弹，不同于我机子弹，敌弹的运行轨迹可以朝着任意一个方向。计算敌弹运行轨迹常用的方法是使用三角函数，根据飞行方向和横坐标轴正方向的夹角来计算敌弹的位移量。虽然 MIDP2.0 提供了三角函数的类库，但是在每个 gameloop 中计算每颗敌弹的运行轨迹势必有大量的三角函数运算而导致性能损失，当子弹数目变大时这种影响更加明显。

同时由于 MIDP1.0 并不提供三角函数类库，所以该游戏无法运行在不支持 MIDP2.0 的手机上。

由基本的数学原理可知，当角度一定时其对应的三角函数值也是一定的，又因为三角函数是周期函数，因此本文的优化算法是用 0°～45°的函数值通过简单的四则运算来模拟任意函数值。具体的实现过程如下。

首先定义如下数组：

```
public static int iAngle[][]=new int[][]{
//根据游戏需要定义不同的精度，可以从 0 连续定义到 45 度。
//因为 SinX=Cos(90-X)，所以不必重复定义 46 度到 90 度的函数值。
  {0，1000，0}，
  {5，996，87}，
  {10，985，174)，
  {40，766，643}，
  }45，707，707}
  }
```

无论子弹朝着哪个方向飞行，假设飞行速度一定，其横纵坐标位移量均为该方向对应的余弦和正弦值，所以可以重新定义敌机子弹数组，将位移量包含到数组中。本游戏设定每颗子弹的飞行方向始终是固定不变的，即不会出现弧形的飞行轨迹。代码如下：

```
  int[][] enemyBullet=new int[4][20];
  //假设同屏同时显示 20 颗敌机子弹
  int size=enemyBultet. length;
  for(int i=0: i
  enemyBullet[0][i]=999; //对应敌机子弹对象的 x 坐标
  enemyBullet[1][i]=999; //对应敌机子弹对象的 y 坐标
  enemyBullet[2][i]=999;
  //对应敌机子弹对象的运行方的 x 坐标位移量
  enemyBullet[3][i]=999;
  //对应敌机子弹对象的运行方的 y 坐标位移量
  }
```

按照程序的需要，从数组 iAngle 中把特定的数值赋值到 enemyBullet[2][i]和 enemyBullet[3][i]，而在每一个 gameloop 中只需要执行如下代码：

```
  inc size=enemyBullet. length:
  for(int i=0; i
   if(enemyBullet[0][i]!=999&&enemyBullet[1][i]!=999){
  enemyBullet[0][i]+=enemyBullet[2][i];
  enemyBullet[1][i]+=enemyBullet[3][i];
  }
  }
```

以上过程避免了每个 gameloop 中使用三角函数计算子弹的位移，同时也使 MIDP1.0 平台的手机不会因为不支持 MIDP2.0 而无法运行程序。为了保证足够的精度，这里相关函数值取 3 位小数，在处理时可以将横纵坐标值 enemyBullet[0][i]和 enemyBullet[1] [i]定义成坐标值的 1000 倍，然后与坐标 值 enemyBullet[2][i]和 enemyBullet[3][i]相加，显示的时候将 enemyBullet[0][i]和 enemyBullet[1][i]的值除以 1000 取整。

当前 J2ME 游戏性能的瓶颈在于有限的存储资源和偏弱的处理器速度，而 J2ME 游戏优化的关键之处在于节省资源开销、节省冗余计算和计算简化，所以牺牲部分设计上的结构化(即面向

对象)换来性能的提升，在当前情况下仍然是非常有必要的。本文提出的优化方法为在较低的硬件条件下实现流畅的画面，提供了可行的解决方案。

4. 找出内存溢出元凶

在编写代码后有时候会抛出 java.lang.OutofMemoryError 异常，就是 java 的内存溢出。笔者曾经上网查了不少资料，试过一些办法，代码也稍微做了些优化，但是有一个问题我始终找不到解决的方案，不知为什么子窗体关闭后 java 的垃圾回收机制无法回收其资源，因为这个 Java 程序可能要经常开关一些子窗体，那么这些子窗体关闭后无法释放资源就造成了 Java 程序 OutOfMemoryError 的潜在隐患。

后来经过前辈指点，事实告诉我之前那个子窗体关闭后资源无法释放的根本原因是：子窗体虽然调用了 dispose()方法，但是子窗体对象的引用一直都在；或者是被静态 HashMap 引用、或者是它的内部子线程类没有释放；或者是它的某个事件监听类没有释放，我们需要彻底释放某个对象占用资源的关键在于找到并释放所有对它的引用。

程序中造成内存溢出可能性最大的是 HashMap、Hashtable 等集合类，尤其是静态的，更是要慎之又慎。它们引用的对象可能你感觉已经销毁了，其实很可能你忘记 remove 键值，而如果这些集合对象还是静态的挂在其他类里面，那么这个引用可能一直都在，借用 JProbe 测试一下，结果往往出人意料，解决办法是彻底删除键 remove、clear，如果允许最好把集合对象设为 null。

对于不再使用的线程对象，如果要彻底杀了它，很多书上都推荐用 join 方法，我之前也是这样做的，但后来借助 JProbe 工具发现这样做很可能要杀的线程仍旧好好地活在你日益增大的内存里，很可能调用了线程的 sleep 方法后用 join 方法就会有点问题，解决办法是在 join 方法前再加一句执行 interrupt 方法，不过这个时候可能会有新的问题：执行 interrupt 方法后你的线程类会抛弃 InterruptedException，上有政策下有对策，加一个开关变量做判断就能完美解决。代码如下：

```
 /**
 * <p>Description: 创建线程的内部类</p>
 * @author cuishen
 * @version 1.1
 */
class NewThread implements Runnable {
   Thread t;
   NewThread() {
      t = new Thread(this, path);
      t.start();
   }
   public void run() {
      try {
         while(isThreadAlive) {
            startMonitor();
            Thread.sleep(Long.parseLong(controlList.get(controlList.size()
            - 1).toString()));
         }
      } catch (InterruptedException e) {
         if(!ifForceInterruptThread) {//开关变量
            stopThread(logThread);
```

```
                String error = "InterruptedException！！！" + path + ": Interrupted,
                线程异常终止！程序已试图重启该线程！！";
                System.err.println(error);
                LogService.writeLog(error);
                createLogThread();
            }
        }
    }
}

public void createLogThread() {
    ifForceInterruptThread = false;//开关变量
    logThread = new NewThread();
}

private void stopThread(NewThread thread) {
    try {
        thread.t.join(100);
    } catch (InterruptedException ex) {
        System.out.println("线程终止异常！！！");
    } finally {
        thread = null;
    }
}
/**
 * 关闭并彻底释放该线程资源的方法
 */
public void stopThread() {
    try {
        ifForceInterruptThread = true;//开关变量
        isThreadAlive = false;
        logThread.t.interrupt();
        logThread.t.join(50);
    } catch (InterruptedException ex) {
        System.out.println("线程终止异常！！！");
    } finally {
        this.controlList = null;
        this.keyList = null;
        logThread = null;
    }
}
```

对于继承 JFrame 的窗体类，我们要注意在初始化方法中加入“this.setDefaultCloseOperation (DISPOSE_ON_CLOSE);”，注意和其关联的事件监听类一律写成窗体类的内部类，这样窗体 dispose()的时候，这些内部类也一并销毁，就不会再有什么莫名其妙的引用了。

注意：JProbe 是一个监控 Java 程序内存使用的工具。

14.4.3 高效 Android

Android 设备是嵌入式设备，即使是“最快”的手持设备，其性能也赶不上一台普通的台式电脑，所以在书写 Android 应用程序的时候要格外关注效率。“这些设备并没有那么快，并且受电池电量的制约”，这意味着，设备没有更多的能力，我们需要做得是必须把程序写得尽量有效。

对于占用资源的系统，有如下两条可遵循的基本原则。

- 不要做不必要的事。
- 不要分配不必要的内存。

微优化(micro-optimization)往往会带来很多的问题，诸如无法使用更有效的数据结构和算法，但是在手持设备上，你别无选择。假如你认为 Android 虚拟机的性能与台式机相当，程序很有可能一开始就占用了系统的全部内存(内存很小)，这会让你的程序慢得像蜗牛一样，更不用说做其他的操作了。

Android 的成功依赖于程序提供的用户体验，而这种用户体验，部分依赖于你的程序是响应快速而灵活的，还是响应缓慢而僵化。因为所有的程序都运行在同一个设备上，都在一起，这就如同在一条路上行驶的汽车。而这篇文档就相当于你在取得驾照之前必须要学习的交通规则。如果大家都按照这些规则去做，驾驶就会很顺畅，但是如果你不这样做，你可能会车毁人亡。这就是为什么这些原则十分重要。

不管 VM 是否支持实时(JIT)编译器(它允许实时地将 Java 解释型程序自动编译成本机机器语言，以使程序执行的速度更快，有些 JVM 包含 JIT 编译器)，下面提到的这些原则都是成立的。假如我们有目标完全相同的两个方法，在解释执行时 foo()比 bar()快，那么编译之后，foo()依然会比 bar()快，所以不要寄希望于编译器可以拯救你的程序，具体的原则如下。

(1) 避免建立对象。

世界上没有免费的对象。虽然 GC 为每个线程都建立了临时对象池，可以使创建对象的代价变得小一些，但是分配内存永远都比不分配内存的代价大。

如果你在用户界面循环中分配对象内存，就会引发周期性的垃圾回收，用户就会觉得界面像打嗝一样一顿一顿的。所以除非必要，应尽量避免建立对象的实例。下面的分析将帮助你理解这条原则。

- 当你从用户输入的数据中截取一段字符串时，尽量使用 substring 函数取得原始数据的一个子串，而不是为子串另外建立一份拷贝。这样你就有一个新的 String 对象，它与原始数据共享一个 Char 数组。
- 如果你有一个函数返回一个 String 对象，而你确切地知道这个字符串会被附加到一个 StringBuffer，那么，请改变这个函数的参数和实现方式，直接把结果附加到 StringBuffer 中，而不要再建立一个短命的临时对象。一个更极端的实例是，把多维数组分成多个一维数组。
- Int 数组比 Integer 数组好，这也概括了一个基本事实，两个平行的 Int 数组比(int,int)对象数组性能要好很多。同理，这适用于所有基本类型的组合。
- 如果你想用一种容器存储(Foo,Bar)元组，尝试使用两个单独的 Foo[]数组和 Bar[]数组，一定比(Foo,Bar)数组效率更高(也有例外的情况，就是当你建立一个 API，让别人调用它的时候。这时候你要注重对 API 接口的设计而牺牲一点儿速度，当然在 API 的内部，

你仍要尽可能地提高代码的效率)。

总体来说，就是避免创建短命的临时对象，减少对象的创建就能减少垃圾收集，进而减少对用户体验的影响。

(2) 使用本地方法。

当在处理字符串的时候，不要吝惜使用 String.indexOf()、String.lastIndexOf()等特殊实现的方法(specialty methods)。这些方法都是使用 C/C++实现的，比起 Java 循环快 10～100 倍。

(3) 使用实类比接口好。

假设你有一个 HashMap 对象，你可以将它声明为 HashMap 或者 Map，具体代码如下：

```
Map myMap1 = new HashMap();
HashMap myMap2 = new HashMap();
```

究竟哪个更好呢？

按照传统的观点，Map 会更好些，因为这样你可以改变它的具体实现类，只要这个类继承自 Map 接口。传统的观点对于传统的程序是正确的，但是它并不适合嵌入式系统，调用一个接口的引用会比调用实体类的引用多花费一倍的时间。

如果 HashMap 完全适合你的程序，那么使用 Map 就没有什么价值。如果有些地方你不能确定，先避免使用 Map，剩下的交给 IDE 提供的重构功能好了(当然公共 API 是一个例外，一个好的 API 常常会牺牲一些性能)。

(4) 用静态方法比虚方法好。

如果你不需要访问一个对象的成员变量，那么请把方法声明成 Static。虚方法执行得更快，因为它可以被直接调用而不需要一个虚函数表。另外，你也可以通过声明体现出这个函数的调用不会改变对象的状态。

(5) 不用 getter 和 setter。

在很多本地语言如 C++中，都会使用 getter(比如：i = getCount())来避免直接访问成员变量(i = mCount)。在 C++中这是一个非常好的习惯，因为编译器能够内联访问，如果你需要约束或调试变量，你可以在任何时候添加代码。

在 Android 中这就不是个好主意了，虚方法的开销比直接访问成员变量大得多。在通用的接口定义中，可以依照 OO 的方式定义 getters 和 setters，但是在一般的类中，你应该直接访问变量。

(6) 将成员变量缓存到本地。

访问成员变量比访问本地变量慢得多，例如下面的一段代码：

```
for (int i = 0; i < this.mCount; i++) dumpItem(this.mItems[i]);
```

最好改成下面这样的代码：

```
int count = this.mCount;
Item[] items = this.mItems;
for (int i = 0; i < count; i++) dumpItems(items[i]); //使用 this 是为了表明这些是
                                                      成员变量
```

另一个相似的原则是：永远不要在 for 的第二个条件中调用任何方法。例如下面的方法，在每次循环的时候都会调用 getCount()方法，这样做比你在一个 Int 先把结果保存起来开销要大很多。例如下面的代码：

```
for (int i = 0; i < this.getCount(); i++) dumpItems(this.getItem(i));
```

同样如果你要多次访问一个变量，也最好先为它建立一个本地变量。例如下面的代码：

```
protected void drawHorizontalScrollBar(Canvas canvas, int width, int height)
{
if (isHorizontalScrollBarEnabled())
{
int size = mScrollBar.getSize(false);
if (size <= 0) { size = mScrollBarSize;
}
mScrollBar.setBounds(0, height - size, width, height);
mScrollBar.setParams( computeHorizontalScrollRange(), computeHorizontalScrollOffset(),
computeHorizontalScrollExtent(), false);
mScrollBar.draw(canvas);
}
}
```

此处有 4 次访问成员变量 mScrollBar，如果将它缓存到本地，4 次成员变量访问就会变成 4 次效率更高的栈变量访问。

另外，方法的参数与本地变量的效率相同。

(7)　使用常量。

让我们先看看这两段在类前面的声明：

```
static int intVal = 42;
static String strVal = "Hello, world! ";
```

这样会生成一个叫做<clinit>的初始化类的方法，当类第一次被使用的时候这个方法会被执行。方法会将 42 赋给 intVal，然后把一个指向类中常量表的引用赋给 strVal。当以后要用到这些值的时候，会在成员变量表中查找到他们。下面我们做些改进，使用“final”关键字，代码如下：

```
static final int intVal = 42;
static final String strVal = "Hello, world!";
```

现在，类不再需要<clinit>方法，因为在成员变量初始化的时候，会将常量直接保存到类文件中。用到 intVal 的代码被直接替换成 42，而使用 strVal 的会指向一个字符串常量，而不是使用成员变量。

将一个方法或类声明为“final”不会带来性能的提升，但是会帮助编译器优化代码。例如，如果编译器知道一个“getter”方法不会被重载，那么编译器会对其采用内联调用。

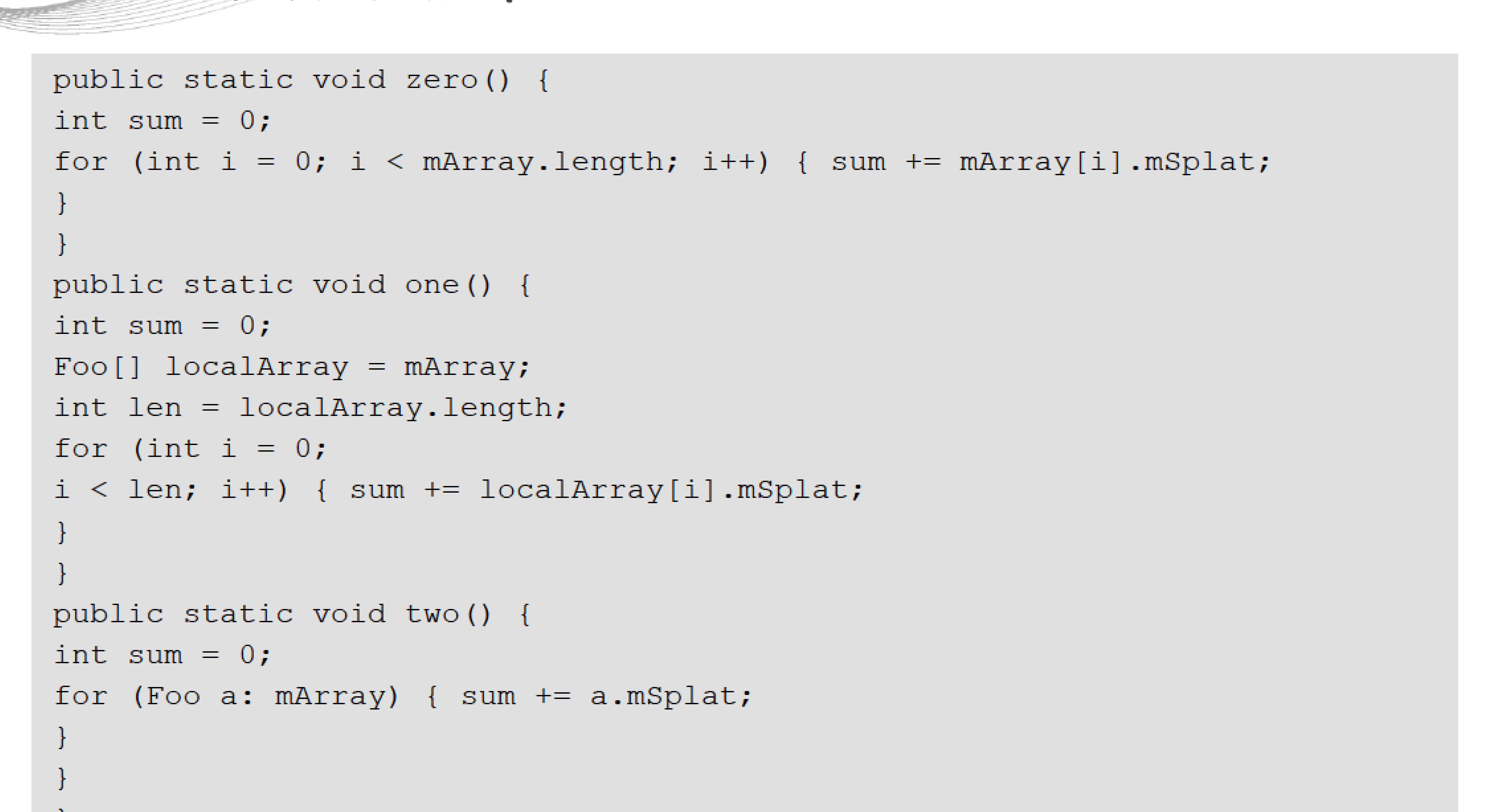
你也可以将本地变量声明为“final”，同样，这也不会带来性能的提升。使用“final”只能使本地变量看起来更清晰(但是也有些时候这是必需的，比如在使用匿名内部类的时候)。

(8)　谨慎使用 foreach。

foreach 可以用在实现了 Iterable 接口的集合类型上。foreach 会给这些对象分配一个 iterator，然后调用 hasNext()和 next()方法。你最好使用 foreach 处理 ArrayList 对象，但是对其他集合对象，foreach 相当于使用 iterator。

下面展示了 foreach 一种可接受的用法，代码如下：

```
public class Foo { int mSplat; static Foo mArray[] = new Foo[27];
```



```
public static void zero() {
int sum = 0;
for (int i = 0; i < mArray.length; i++) { sum += mArray[i].mSplat;
}
}
public static void one() {
int sum = 0;
Foo[] localArray = mArray;
int len = localArray.length;
for (int i = 0;
i < len; i++) { sum += localArray[i].mSplat;
}
}
public static void two() {
int sum = 0;
for (Foo a: mArray) { sum += a.mSplat;
}
}
}
```

对上述代码进行如下分析。

- 在 zero()中，每次循环都会访问两次静态成员变量，取得一次数组的长度。
- 在 one()中，将所有成员变量存储到本地变量。
- 在 two()中，使用了在 java1.5 中引入的 foreach 语法。编译器会将对数组的引用和数组的长度保存到本地变量中，这对访问数组元素非常好。但是编译器还会在每次循环中产生一个额外的对本地变量的存储操作(对变量 a 的存取)，这样会比 one()多出 4 个字节，速度要稍微慢一些。

综上所述，foreach 语法在运用于 Array 时性能很好，但是运用于其他集合对象时要小心，因为它会产生额外的对象。

(9) 避免使用枚举。

枚举变量非常方便，但不幸的是它会牺牲执行的速度并大幅增加文件体积。例如下面的代码：

```
public class Foo {
public enum Shrubbery {
GROUND, CRAWLING, HANGING
}
}
```

会产生一个 900 字节的.class 文件(Foo$Shubbery.class)。在它被首次调用时，这个类会调用初始化方法来准备每个枚举变量。每个枚举项都会被声明成一个静态变量并被赋值。然后将这些静态变量放在一个名为“$VALUES”的静态数组变量中。而这么一大堆代码，仅仅是为了使用三个整数。如果采用下面的代码：

```
Shrubbery shrub = Shrubbery.GROUND;
```

会引起一个对静态变量的引用，如果这个静态变量是 final int，那么编译器会直接内联这个常数。

使用枚举变量可以让你的 API 更出色，并能提供编译时的检查。所以在通常的情况下，你毫

无疑问应该为公共 API 选择枚举变量，但是当性能方面有所限制的时候，你就应该避免这种做法了。

在有些情况下，使用 ordinal()方法获取枚举变量的整数值会更好一些。例如下面的代码：

```
for (int n = 0; n < list.size(); n++) { if (list.items[n].e == MyEnum.VAL_X) //
do stuff1 else if (list.items[n].e == MyEnum.VAL_Y) // do stuff 2 }
```

替换为下面的代码：

```
int valX = MyEnum.VAL_X.ordinal();
int valY = MyEnum.VAL_Y.ordinal(); int count = list.size();
MyItem items = list.items();
for (int n = 0; n < count; n++) { int valItem = items[n].e.ordinal();
if (valItem == valX) // do stuff 1
else if (valItem == valY) // do stuff 2
}
```

这样会使性能得到一些改善，但这并不是最终的解决之道。将与内部类一同使用的变量声明在包范围内，例如下面的类定义：

```
public class Foo {
private int mValue;
public void run() {
Inner in = new Inner();
mValue = 27; in.stuff();
}
private void doStuff(int value) {
System.out.println("Value is " + value); }
private class Inner {
void stuff() {
Foo.this.doStuff(Foo.this.mValue);
}
}
}
```

这其中的关键是我们定义了一个内部类(Foo$Inner)，它需要访问外部类的私有域变量和函数，这是合法的并且会打印出我们希望的结果“Value is 27”。问题是在技术上来说(在幕后)Foo$Inner 是一个完全独立的类，它要直接访问 Foo 的私有成员是非法的，要跨越这个鸿沟，编译器需要生成一组方法。代码如下：

```
static int Foo.access$100(Foo foo) {
return foo.mValue;
}
static void Foo.access$200(Foo foo, int value) {
foo.doStuff(value);
}
```

内部类在每次访问 mValue 和 doStuff 方法时，都会调用这些静态方法。就是说，上面的代码说明了一个问题，你是在通过接口方法访问这些成员变量和函数而不是直接调用它们。在前面我们已经说过，使用接口方法(getter、setter)比直接访问速度要慢。所以，这个例子就是在特定语法下产生的一个“隐性的”性能障碍。

通过将内部类访问的变量和函数声明由私有范围改为包范围，可以避免这个问题。这样做可以让代码运行更快，并且避免产生额外的静态方法(遗憾的是，这些域和方法可以被同一个包内的其他类直接访问，这与经典的 OO 原则相违背。因此当你设计公共 API 的时候应该谨慎使用这条优化原则)。

(10) 避免使用浮点数。

在奔腾 CPU 出现之前，游戏设计者做得最多的就是整数运算。随着奔腾的到来，浮点运算处理器成了 CPU 内置的特性，浮点和整数配合使用，能够让你的游戏运行得更顺畅。通常在桌面电脑上，你可以随意地使用浮点运算。

但是非常遗憾，嵌入式处理器通常没有支持浮点运算的硬件，所有对 float 和 double 的运算都是通过软件实现的。一些基本的浮点运算，甚至需要毫秒级的时间才能完成。甚至是整数，一些芯片有对乘法的硬件支持但缺少对除法的支持。这种情况下，整数的除法和取模运算也是由软件来完成的。所以，当你在使用哈希表或者做大量数学运算时务必要小心谨慎。

14.4.4 Android 的单元测试

任何程序的开发都离不开单元测试来保证其健壮和稳定。Android 的程序自然也不例外。从 Android SDK 0.9 开始，就有了比较成熟的测试框架。接下来，笔者在这里对此内容做一下梳理和总结。

(1) 谈谈 JUnit。

在 Java 平台下做单元测试必然用到 JUnit。这里说的 JUnit 是指从 Apache 基金会下载的 junit.jar 里提供的一系列单元测试功能。这些功能显然是运行在 JDK 之上的。在 Android 下已经没有了 JDK，自然也无法运行 JUnit，但是这并不妨碍我们利用 JUnit 编写单元测试。只不过在运行单元测试时，一定要用 JDK 来运行，利用 Java 命令来启动 JUnit 的某个 Runner。如果是用 Eclipse 的话，可以在 Run Configuration 里新建一个 JUnit。但是一定要记得在 Classpath 选项卡里将 Bootstrap Entries 中的 Android Library 改成 JRE，并且添加 junit.jar，如图 14-1 所示。

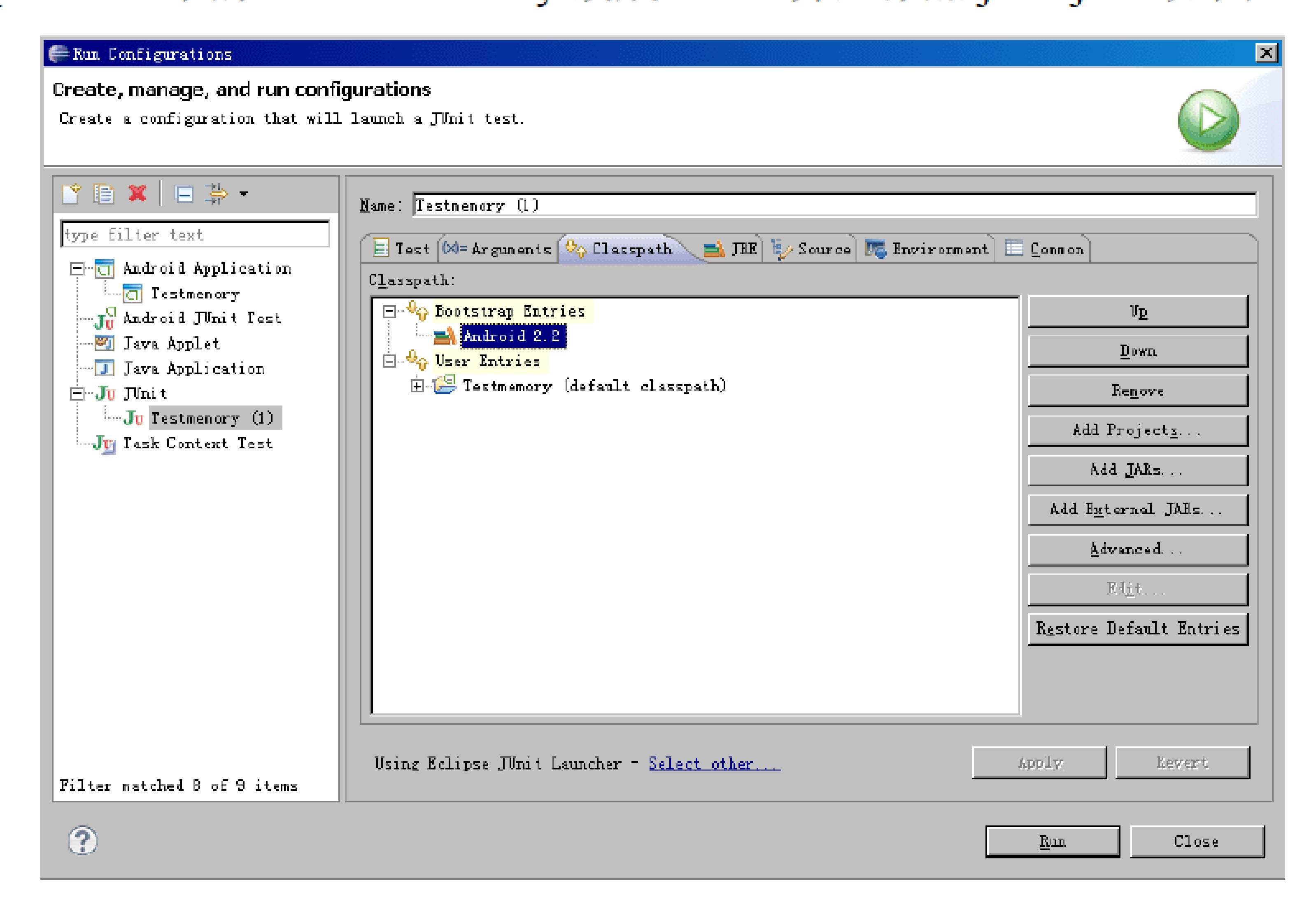


图 14-1　JUnit

很明显，这种测试就是正规的 Java 单元测试，同 Android 没有任何关系。你无法测试任何关于 Android 系统中的 API，例如，Activity、人机界面等。所以，如果你想测试的仅仅是一些封装数据的对象，或者是纯粹的数值计算，还是可以用这种方法的。

(2)　junit.framework。

当读者看到这个包的时候，第一反应是 Android 是不是已经完整集成了 JUnit，很遗憾这不是事实。如果你按照 JUnit 的运行方法，却不像上面那样改用 JDK，就一定会得到如下的异常：

```
#
# An unexpected error has been detected by Java Runtime Environment:
#
# Internal Error (classFileParser.cpp:2924), pid=4900, tid=4476
#Error: ShouldNotReachHere()
#
# Java VM: Java HotSpot(TM) Client VM (10.0-b19 mixed mode windows-x86)
# An error report file with more information is saved as:
# E:\Mydoc\EclipseWorkspace\TestAndroid\hs_err_pid4900.log
#
# If you would like to submit a bug report, please visit:
# http://java.sun.com/webapps/bugreport/crash.jsp
#
```

实际上，TestCase 类用于在 Android 中担当所有独特的 TestCase 的基类作用，它是一个 Abstract Class，里面有很多以 TestCase 为后缀的类。之所以有那么多 TestCase 主要是为了简化工作。例如，当你想对一个访问数据库的功能进行测试时，首先需要自己启动并初始化数据库。在这里是类似的，如果你想测试一个 Activity，首先要启动它。而 ActivityTestCase 就会自动帮助你做完这些事情。而 ActivityUnitTestCase 会更注重测试的独立性，它会让测试与 Android 底层的联系降到最低，其余的类可以查看相关的 Javadoc 来按需挑选。要编写测试，就要找到合适的 XXXTestCase 作为基类来继承，并且编写自己的测试方法。

很明显，最简单的编写测试的方法就是继承 AndroidTestCase 写一个自己的 TestCase。然后，为自己的一组 TestCase 写一个 Activity 界面，由界面控制 TestCase 的启动、运行和结果报告。但是，你很快会发现，为何要给测试写一个界面呢？这太诡异了，这时就需要一种技术，它可以利用命令行(Shell)来启动一组测试，并且通过命令行的形式给出结果，这就是所谓的 Instrumentation。

(3)　Instrumentation。

一般在开发 Android 程序的时候，需要写一个 manifest 文件，其代码如下：

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
<activity android:name=".TestApp" android:label="@string/app_name">
…
</activity>
</application>
```

在启动程序的时候就会先启动一个 Application，然后在 Application 运行过程中根据情况加载相应的 Activity，而 Activity 是需要一个界面的，但是 Instrumentation 并不是这样的。你可以将 Instrumentation 理解为一种没有图形界面，具有启动能力，用于监控其他类(用 Target Package 声明)的工具类。任何想成为 Instrumentation 的类必须继承 android.app.Instrumentation。

对于单元测试，我们需要认真了解的就是 android.test.InstrumentationTestRunner 类。这是 Android 单元测试的主入口。它相当于 JUnit 中 TestRunner 的作用。

那么如何加载它呢？首先要在 manifest 文件中加入一行关于 Instrumentation 的声明。例如 Android Api Demos 测试中的 manifest 如下：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.android.apis.tests">
<application>
<uses-library android:name="android.test.runner" />
</application>
<instrumentation android:name="android.test.InstrumentationTestRunner"
android:targetPackage="com.example.android.apis"
android:label="Tests for Api Demos. "/>
</manifest>
```

当编辑好 manifest 就可以打包(build，可以用 Eclipse ADT 来做，也可以用 aapt 命令手工完成)，然后安装到虚拟机上(用 adb install 命令)。之后就可以利用命令行的方式来加载你的单元测试了。在 Android Shell 中加载一个 Instrumentation 的方法是利用下面的命令：

```
adb shell am instrument -w XXXXXX
```

其中-w 是指定 Instrumentation 类的参数标志。例如下面的一个简单例子：

```
adb shell am instrument -w com.android.foo/android.test.InstrumentationTestRunner
```

当然，也可以利用 adb shell 先进入 android 命令行模式，再直接写 am instrument －w XXXXXXX。下面将具体介绍如何根据需要加载一组单元测试。

(4) 如何在 Android 中利用 Instrumentation 来进行测试。

在介绍具体的命令之前，我们先理解一下单元测试的层次。一组单元测试可以被组织成若干个 TestSuite。每个 TestSuite 包含若干 TestCase(某个继承 android.jar 的 junit.framework.TestCase 类)。每个 TestCase 又包含若干个 Test(具体的 test 方法)。

如果假设 com.android.foo 是你的测试代码包的根。当执行以下命令时，会执行 TestCase 的所有 Test。测试的对象就是在 Target Package 中指定包中的代码：

```
adb shell am instrument -w com.android.foo/android.test.InstrumentationTestRunner
```

如果你想运行一个 TestSuite，首先继承 android.jar 的 junit.framework.TestSuite 类，实现一个 TestSuite(例如叫 com.android.foo.MyTestSuite)，然后执行以下命令执行此 TestSuite：

```
adb shell am instrument -e class com.android.foo.MyTestSuite -w com.android.foo/
android.test.InstrumentationTestRunner
```

其中-e 表示额外的参数，语法格式如下：

```
-e [arg1] [value1] [arg2] [value2]
```

如果仅仅想运行一个 TestCase(例如叫 com.android.foo.MyTestCase)，在此用到了 class 参数。则用以下的命令：

```
adb shell am instrument -e class com.android.foo.MyTestCase -w com.android.foo/
android.test.InstrumentationTestRunner
```

如果仅仅想运行一个 Test(例如上面 MyTestCase 的 testFoo 方法)，可以这样写代码：

```
adb shell am instrument -e class com.android.foo.MyTestCase#testFoo -w com.android.foo/
android.test.InstrumentationTestRunner
```

所有的测试结果会输出到控制台，并且会做一系列统计，如标记为 E 的是 Error，标记为 F 的是 Failure，Success 的测试则会标记为一个点。这和 JUnit 的语义一致。如果希望断点调试你的测试，只需要直接在代码上加上断点，然后将运行命令参数的-e 后边附加上 debug true 后运行即可。更加详细的内容可以看 InstrumentationTestRunner 的 Javadoc。

(5)　如何在 Android 的单元测试中做标记。

在 android.test.annotation 包里定义了几个 annotation，包括@LargeTest、@MediumTest、@SmallTest、@Smoke 和@Suppress。你可以根据自己的需要用这些 annotation 来对自己的测试分类。在执行单元测试命令时，可以在-e 参数后设置"size large"/"size medium"/"size small"来执行具有相应标记的测试。特别的@Suppers 可以取消被标记的 Test 执行。

14.5　UI 界面优化

很多程序员急功近利，为了实现某些功能，只要编写出对应的代码即可，也不考虑具体的性能。但这样对我们追求精益求精的思想是背道而驰的，往往就是因为满足于一个结果，而放弃探求更加优化的处理方法。

当关注应用程序或者游戏所达到的结果时，往往非常容易忽视一些优化的问题，例如，内存优化、线程优化、Media 优化和 UI 优化等。不同的模块都存在更为巧妙的方式来对待一般性问题，所以每当我们实现一个行为后，稍微多花一些时间来考虑目前所做的工作是否存在更为高效的解决办法是很有必要的。

在 Android 中，LinearLayout 表示 UI 的框架，而且也是最直观和方便的方法。例如，创建一个 UI 用于展现 Item 的基本内容，如图 14-2 所示。将图形用一个线框形式表示，如图 14-3 所示。

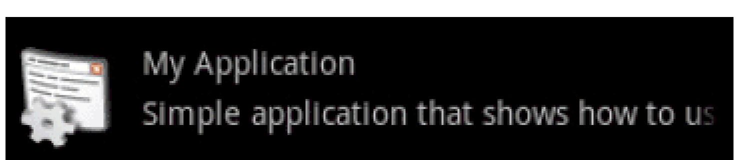


图 14-2　LinearLayout 布局



图 14-3　线框图

可以通过 LinearLayout 来快速实现这个 UI 的排列。主要代码如下：

```
<LinearLayout xmlns:
    android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="6dip">
    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_marginRight="6dip"
        android:src="@drawable/icon" />
```

```
    <LinearLayout
    android:orientation="vertical"
    android:layout_width="0dip"
    android:layout weight="1"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="0dip"
        android:layout_weight="1"
        android:gravity="center_vertical"
        android:text="My Application" />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="0dip"
        android:layout_weight="1"
        android:singleLine="true"
        android:ellipsize="marquee"
        android:text="Simple application that shows how to use RelativeLayout" />
    </LinearLayout>
</LinearLayout>
```

虽然可以通过 Linearlayout 实现我们所预想的结果，但是在这里存在一个优化的问题，尤其是针对大量 Items。比较 RelativeLayout 和 LinearLayout，在资源利用上前者会占用更少的资源而达到相同的效果，下面是用 RelativeLayout 实现的代码：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="6dip">

    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_alignParentTop="true"
        android:layout_alignParentBottom="true"
        android:layout_marginRight="6dip"

        android:src="@drawable/icon" />

    <TextView
        android:id="@+id/secondLine"
        android:layout_width="fill_parent"
        android:layout_height="26dip"
        android:layout_toRightOf="@id/icon"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:singleLine="true"
        android:ellipsize="marquee"
        android:text="Simple application that shows how to use RelativeLayout" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

```
        android:layout_toRightOf="@id/icon"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:layout above="@id/secondLine"
        android:layout_alignWithParentIfMissing="true"
        android:gravity="center_vertical"
        android:text="My Application" />
</RelativeLayout>
```

针对 RelativeLayout 有一点需要注意，因为它内部是通过多个 View 之间的关系而确定的框架，那么当其中某一个 View 因为某些需要调用 GONE 来完全隐藏掉后，会影响与其相关联的 Views。Android 为我们提供了一个属性 alignWithParentIfMissing 用于解决类似问题，当某一个 View 无法找到与其相关联的 Views 后将依据 alignWithParentIfMissing 的设定判断是否与父级 View 对齐。

定义 Android Layout(XML)时，有四个比较特别的标签是非常重要的，其中有三个是与资源复用有关，分别是<viewStub/>、<requestFocus/>、<merge/>和<include/>。可是以往我们所接触的案例或者官方文档的例子都没有着重去介绍这些标签的重要性，下面具体介绍一下。

(1) <viewStub />: 此标签可以使 UI 在特殊情况下，直观效果类似于设置 View 的不可见性，但是其更大的(R)意义在于被这个标签所包裹的 Views 在默认状态下不会占用任何内存空间，viewStub 通过 include 从外部导入 Views 元素。

其用法是通过 android:layout 来指定所包含的内容。默认情况下，ViewStub 所包含的标签都属于 visibility=GONE。viewStub 通过方法 inflate()来召唤系统加载其内部的 Views。例如下面的代码：

```
<ViewStub android:id="@+id/stub"
android:inflatedId="@+id/subTree"
android:layout="@layout/mySubTree"
android:layout_width="120dip"
android:layout_height="40dip" />
```

(2) <merge />：通过删减多余或额外的层级，达到优化整个 Android Layout 结构的目的。

(3) <include/>：可以通过这个标签直接加载外部的 xml 到当前结构中，是复用 UI 资源的常用标签。其用法是将需要复用 xml 文件路径赋予 include 标签的 Layout 属性。例如下面的代码：

```
<include android:id="@+id/cell1" layout="@layout/ar01" />
<include android:layout_width="fill_parent" layout="@layout/ar02" />
```

(4) <requestFocus />：标签用于指定屏幕内的焦点 View，其用法是将标签置于 Views 标签内部。例如下面的代码：

```
<EditText id="@+id/text"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="0"
      android:paddingBottom="4">
   <requestFocus />
</EditText>
```

单独将<merge />标签做个介绍，是因为它在优化 UI 结构时起到很重要的作用。目的是通过删减多余或者额外的层级，从而优化整个 Android Layout 的结构。下面通过实例进行讲解。

题 目	目 的	源码路径
题目 1	演示 UI 布局优化，演示<merge />标签实际所产生的作用	"光盘:\daima\14\Examples"文件夹

(1) 建立一个简单的 Layout，其中包含两个 Views 元素：ImageView 和 TextView。默认状态下我们将这两个元素放在 FrameLayout 中。其效果是在主视图中全屏显示一张图片，之后将标题显示在图片上并位于视图的下方。实现文件 main.xml 的具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <ImageView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:scaleType="center"
        android:src="@drawable/golden_gate"
        />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="20dip"
        android:layout_gravity="center_horizontal|bottom"
        android:padding="12dip"
        android:background="#AA000000"
        android:textColor="#ffffffff"
        android:text="Golden Gate"
        />
</FrameLayout>
```

此时执行后的效果如图 14-4 所示。

(2) 启动 SDK 目录下的"tools"文件夹中的 hierarchyviewer.bat，如图 14-5 所示。

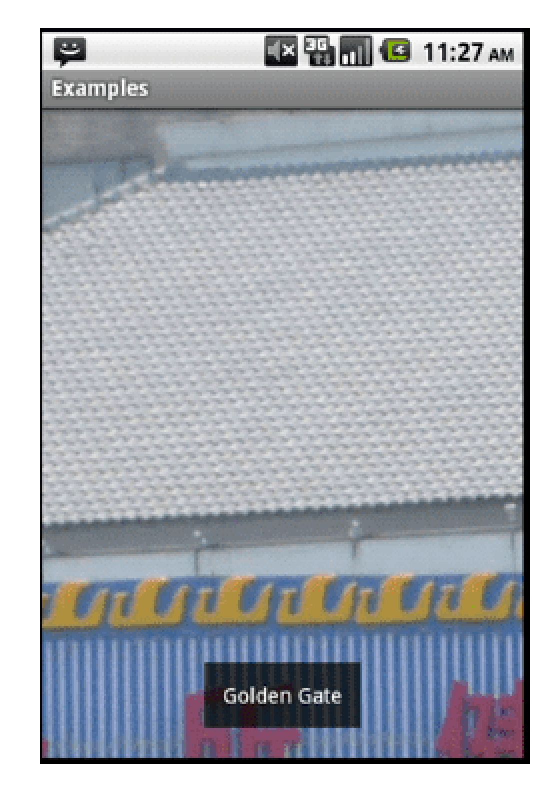


图 14-4 执行效果



图 14-5 启动 hierarchyviewer.bat

此时可以查看当前 UI 的结构视图，如图 14-6 所示。

图 14-6　文件 main.xml 的 UI 结构视图

此时可以很明显地看到由红色线框所包含的结构出现了两个 framelayout 节点，说明这两个完全意义相同的节点造成了资源浪费，那么如何才能解决呢？这时候就要用到<merge />标签来处理类似的问题了。

(3)　将上边 xml 代码中的 framLayout 换成 merge，实现文件 main2.xml 的具体代码如下所示：

```
<merge
    xmlns:android="http://schemas.android.com/apk/res/android"
    >
    <ImageView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:scaleType="center"
        android:src="@drawable/golden_gate"
        />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="20dip"
        android:layout_gravity="center_horizontal|bottom"
        android:padding="12dip"
        android:background="#AA000000"
        android:textColor="#ffffffff"
        android:text="Golden Gate"
```

```
        />
</merge>
```

程序运行后，在 Emulator 中显示的效果是一样的，可是通过 hierarchyviewer 查看到 UI 结构视图是有变化的，如图 14-7 所示。

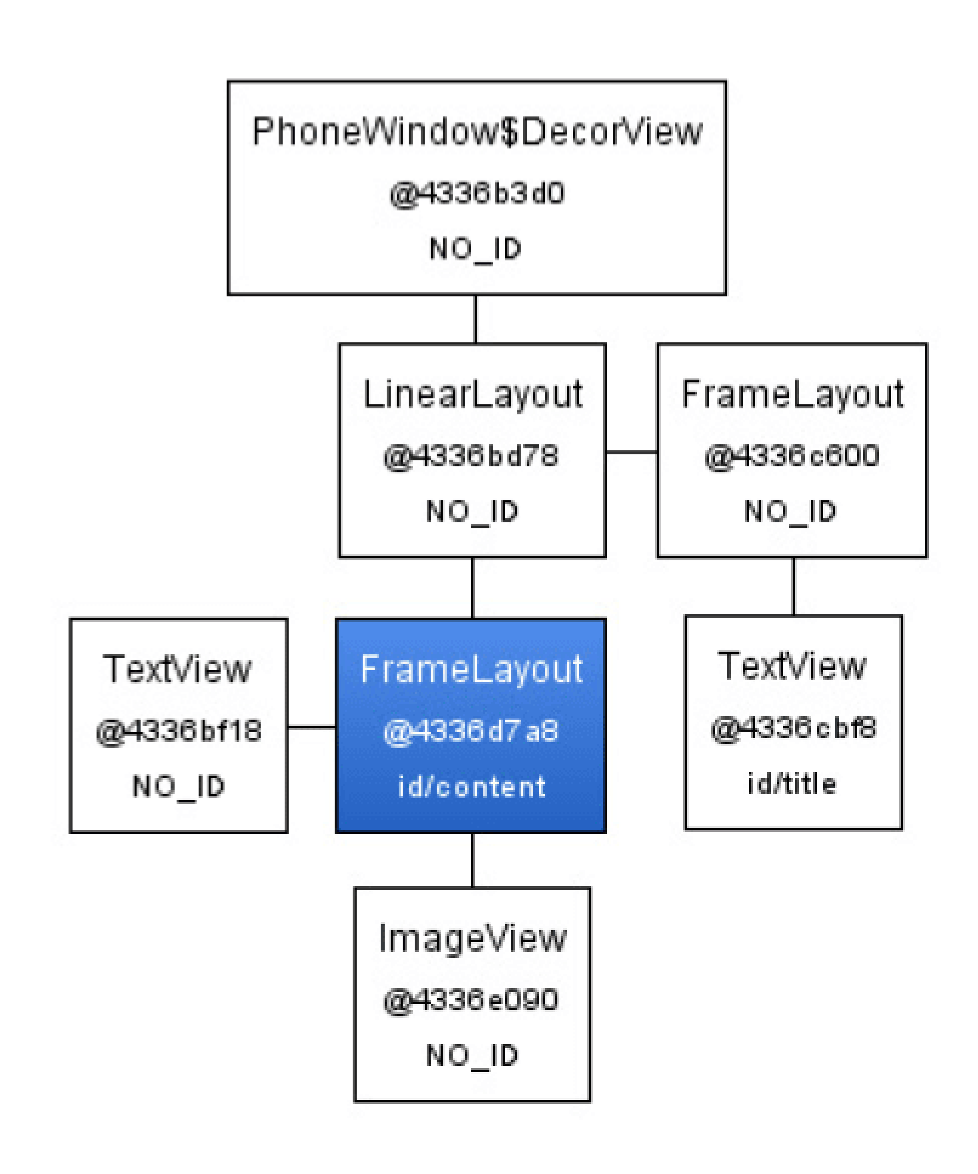


图 14-7　UI 结构视图

此时原来多余的 FrameLayout 节点被合并在了一起，即将 merge 标签中的子集直接加到 Activity 的 FrameLayout 根节点下。如果所创建的 Layout 并不是用 framLayout 作为根节点(而是应用 LinerLayout 等定义 root 标签)，就不能应用上边的例子通过 merge 来优化 UI 结构。

其实除了上面的实例外，meger 还有另外一个用法。当应用 Include 或者 ViewStub 标签从外部导入 xml 结构时，可以将被导入的 xml 用 merge 作为根节点表示，这样当被嵌入父级结构中后可以很好地将它所包含的子集融合到父级结构中，而不会出现冗余的节点。

注意： (1) <merge />只可以作为 xml layout 的根节点。

(2) 当需要扩充的 xml layout 本身是由 merge 作为根节点的话，需要将被导入的 xml layout 置于 viewGroup 中，同时需要设置 attachToRoot 的属性为 True。



第 15 章　Graphics 的魅力

除了拨打电话和发送短信外，智能手机一般还具备音/视频播放、移动上网、蓝牙、收音机、软件下载，游戏等功能。特别是游戏功能，直接影响了手机的销量。在本章的内容中，将详细讲解 Android 手机游戏开发的基础知识——绘图处理。

15.1　绘 图 处 理

看本章的标题就应该知道，Graphics 是和绘图有关的一个功能。在 Android 应用中，可以使用 Graphics 接口绘制各种各样的图形。在本章的内容中，将详细讲解 Android 手机游戏的基础知识——绘图处理。

15.1.1　Color 类

Color 类即 Android.Graphics.Color，在 Android 平台上表示颜色的方法有很多种，Color 提供了常规颜色的定义，比如 Color.BLACK 和 Color.GREEN 等，创建时主要使用以下静态方法。

(1)　static int argb(int alpha, int red, int green, int blue)：构造一个包含透明对象的颜色。

(2)　static int rgb(int red, int green, int blue)：构造一个标准的颜色对象。

(3)　static int parseColor(String colorString)：解析一种颜色字符串的值，比如传入 Color.BLACK。

本类返回的值均为一个整形，类似绿色为 0xff00ff00，红色为 0xffff0000 的值。可以将这个 DWORD 型看做 AARRGGBB，AA 代表 Aphla 透明色，后面的就不难理解，每个分成 WORD 正好为 0～255。

15.1.2　Paint 类

Paint 类即 Android.Graphics.Paint，Paint 类我们可以理解为画笔、画刷的属性定义，本类中常用的方法如下。

(1)　void reset()：重置。

(2) void setARGB(int a, int r, int g, int b)或 void setColor(int color)：均为设置 Paint 对象的颜色。

(3) void setAntiAlias(boolean aa)：是否抗锯齿，需要配合 void setFlags (Paint.ANTI_ALIAS_FLAG)来帮助消除锯齿使其边缘更平滑。

(4) Shader setShader(Shader shader)：设置阴影，Shader 类是一个矩阵对象，如果为 NULL 将清除阴影。

(5) void setStyle(Paint.Style style)：设置样式，一般为 FILL 填充，或者 STROKE 凹陷效果。

(6) void setTextSize(float textSize)：设置字体大小。

(7) void setTextAlign(Paint.Align align)：文本对齐方式。

(8) Typeface setTypeface(Typeface typeface)：设置字体，通过 Typeface 可以加载 Android 内部的字体，对于中文一般为宋体，部分 ROM 可以自己添加，例如，雅黑等。

(9) void setUnderlineText(boolean underlineText)：是否设置下划线，需要结合 void setFlags (Paint.UNDERLINE_TEXT_FLAG)方法一起使用。

题　目	目　的	源码路径
演练 1	用 Color 类和 Paint 类实现绘图处理	“光盘:\daima\15\example1”文件夹

用 color 类和 paint 类实现绘图处理的方法如下。

(1) 编写布局文件 main.xml。具体代码如下所示。

```
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
   Android:orientation="vertical"
   Android:layout_width="fill_parent"
   Android:layout_height="fill_parent"
   >
<TextView
   Android:layout width="fill parent"
   Android:layout_height="wrap_content"
   Android:text="@string/hello"
   />
</LinearLayout>
```

(2) 编写文件 Activity.java，通过 mGameView = new GameView(this)，用 Activity 类的 setContentView 方法来设置要显示的具体 View 类。文件 Activity.java 的主要代码如下所示：

```
public class Activity01 extends Activity
{
    private GameView mGameView;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        mGameView = new GameView(this);
        setContentView(mGameView);
    }
}
```

(3) 编写主文件 draw.java，其功能是绘制出指定的图形。首先声明 Paint 对象 mPaint，定义

draw 分别用于构建对象和开启线程。具体代码如下所示：

```
/* 声明 Paint 对象 */
private Paint mPaint    = null;
public draw(Context context)
{
    super(context);
    /* 构建对象 */
    mPaint = new Paint();
    /* 开启线程 */
    new Thread(this).start();
}
```

然后定义方法 onDraw，先设置 Paint 格式和颜色并根据提取的颜色、尺寸、风格、字体和属性实现绘制处理。具体代码如下所示：

```
public void onDraw(Canvas canvas)
{
    super.onDraw(canvas);
    /* 设置 Paint 为无锯齿 */
    mPaint.setAntiAlias(true);
    /* 设置 Paint 的颜色 */
    mPaint.setColor(Color.WHITE);
    mPaint.setColor(Color.BLUE);
    mPaint.setColor(Color.YELLOW);
    mPaint.setColor(Color.GREEN);
    /* 同样是设置颜色 */
    mPaint.setColor(Color.rgb(255, 0, 0));
    /* 提取颜色 */
    Color.red(0xcccccc);
    Color.green(0xcccccc);
    /* 设置 paint 的颜色和 Alpha 值(a,r,g,b) */
    mPaint.setARGB(255, 255, 0, 0);
    /* 设置 paint 的 Alpha 值 */
    mPaint.setAlpha(220);
    /* 这里可以设置为另外一个 paint 对象 */
    // mPaint.set(new Paint());
    /* 设置字体的尺寸 */
    mPaint.setTextSize(14);
    // 设置 paint 的风格为“空心”
    // 当然也可以设置为“实心”(Paint.Style.FILL)
    mPaint.setStyle(Paint.Style.STROKE);
    // 设置“空心”的外框的宽度
    mPaint.setStrokeWidth(5);
    /* 得到 Paint 的一些属性 */
    Log.i(TAG, "paint 的颜色: " + mPaint.getColor());
    Log.i(TAG, "paint 的 Alpha: " + mPaint.getAlpha());
    Log.i(TAG, "paint 的外框的宽度: " + mPaint.getStrokeWidth());
    Log.i(TAG, "paint 的字体尺寸: " + mPaint.getTextSize());
    /* 绘制一个矩形 */
    // 肯定是一个空心的矩形
    canvas.drawRect((320 - 80) / 2, 20, (320 - 80) / 2 + 80, 20 + 40, mPaint);
    /* 设置风格为实心 */
    mPaint.setStyle(Paint.Style.FILL);
    mPaint.setColor(Color.GREEN);
    /* 绘制绿色实心矩形 */
```

```
        canvas.drawRect(0, 20, 40, 20 + 40, mPaint);
    }
```

最后定义触笔事件 onTouchEvent，按下事件 onKeyDown 的按键，弹起事件 onKeyUp 的按键。主要代码如下所示：

```
    // 触笔事件
    public boolean onTouchEvent(MotionEvent event)
    {
        return true;
    }

    // 按键按下事件
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        return true;
    }

    // 按键弹起事件
    public boolean onKeyUp(int keyCode, KeyEvent event)
    {
        return false;
    }

    public boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event)
    {
        return true;
    }
    public void run()
    {
        while (!Thread.currentThread().isInterrupted())
        {
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
            // 使用 postInvalidate 可以直接在线程中更新界面
            postInvalidate();
        }
    }
}
```

至此，整个演练结束，执行后的效果如图 15-1 所示。

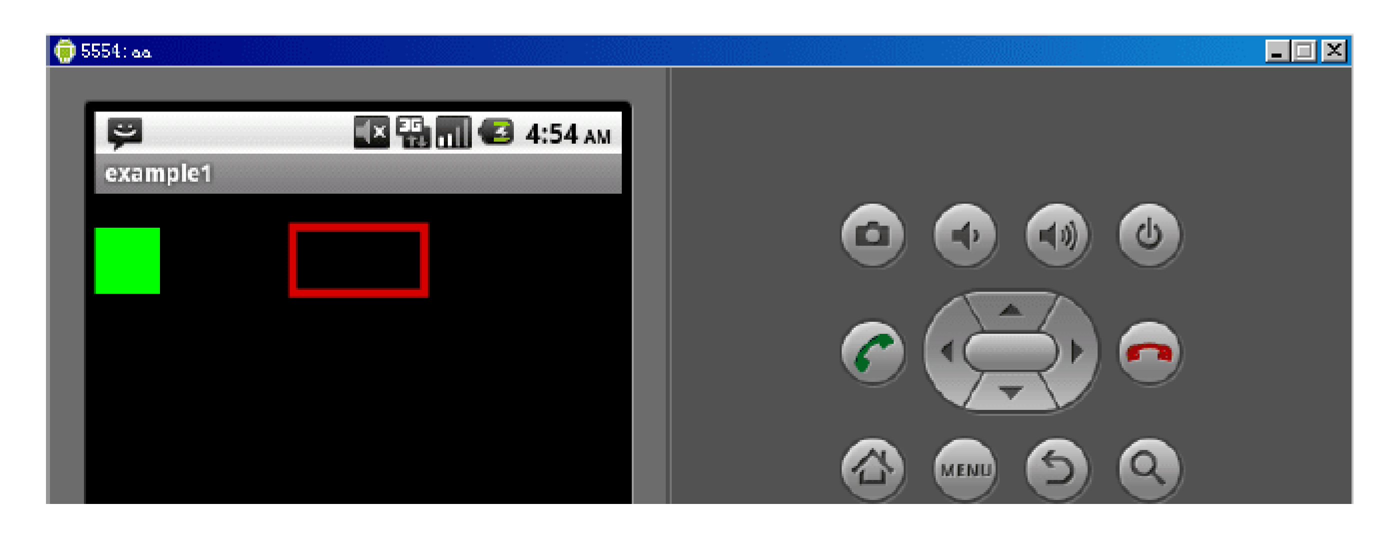


图 15-1　执行效果

15.1.3　Canvas 类

Canvas 即画布，可以将其看作是一种处理过程，使用各种方法来管理 Bitmap、GL 或者 Path 路径，同时它可以配合 Matrix 矩阵类给图像做旋转、缩放等操作，同时 Canvas 类还提供了裁剪、选取等操作。

题　目	目　的	源码路径
演练 2	在 Android 中使用 Canvas 类	"光盘:\daima\15\example2" 文件夹

在本实例中，通过 OnDraw()方法实现了图形绘制功能，将指定图形绘制在了用 Canvas 实现的画布上。主文件 example2.java 的主要代码如下所示：

```
/* 声明 Paint 对象 */
private Paint mPaint   = null;
public example2(Context context)
{
    super(context);
    /* 构建对象 */
    mPaint = new Paint();
    /* 开启线程 */
    new Thread(this).start();
}

public void onDraw(Canvas canvas)
{
    super.onDraw(canvas);

    /* 设置画布的颜色 */
    canvas.drawColor(Color.BLACK);
    /* 设置取消锯齿效果 */
    mPaint.setAntiAlias(true);
    /* 设置裁剪区域 */
    canvas.clipRect(10, 10, 280, 260);
    /* 线锁定画布 */
    canvas.save();
    /* 旋转画布 */
    canvas.rotate(45.0f);
    /* 设置颜色及绘制矩形 */
    mPaint.setColor(Color.RED);
    canvas.drawRect(new Rect(15,15,140,70), mPaint);
    /* 解除画布的锁定 */
    canvas.restore();
    /* 设置颜色及绘制另一个矩形 */
    mPaint.setColor(Color.GREEN);
    canvas.drawRect(new Rect(150,75,260,120), mPaint);
}

// 触笔事件
public boolean onTouchEvent(MotionEvent event)
{
```

```
        return true;
    }
    // 按键按下事件
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        return true;
    }
    // 按键弹起事件
    public boolean onKeyUp(int keyCode, KeyEvent event)
    {
        return false;
    }
    public boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event)
    {
        return true;
    }
    public void run()
    {
        while (!Thread.currentThread().isInterrupted())
        {
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
            // 使用 postInvalidate 可以直接在线程中更新界面
            postInvalidate();
        }
    }
}
```

至此整个演练结束，执行后的效果如图 15-2 所示。

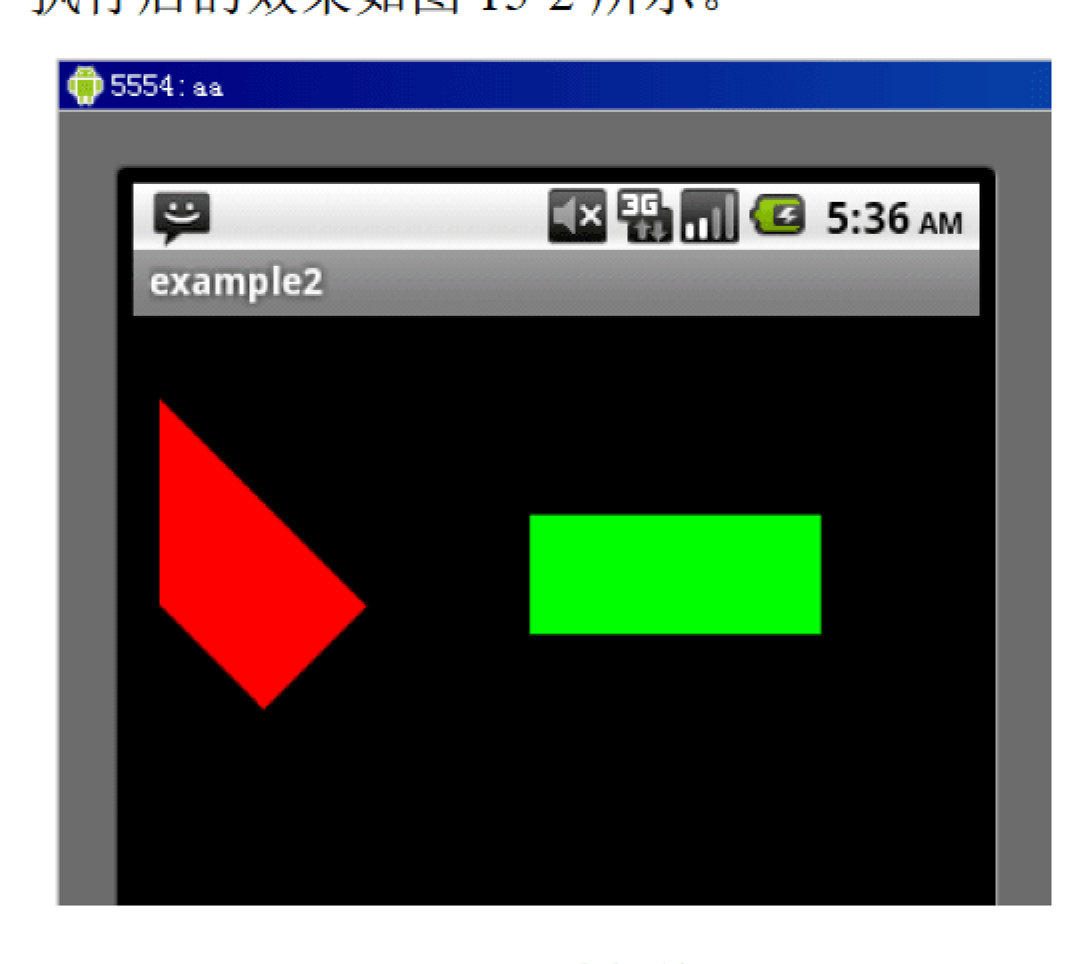


图 15-2　执行效果

15.1.4　Rect 类

Rect 类即 Android.Graphics.Rect，即矩形区域。Rect 类除了表示一个矩形区域位置描述外，还可以帮助计算图形之间是否碰撞(包含)的关系，对于 Android 游戏开发比较有用，其主要的成员 contains 包含了如下 3 种重载方法来判断包含关系。主要代码如下：

```
boolean contains(int left, int top, int right, int bottom)
boolean contains(int x, int y)
boolean contains(Rect r)
```

题　目	目　的	源码路径
演练 3	在 Android 中使用 Rect 类	“光盘:\daima\15\example3”文件夹

在本实例中，首选构建了 Paint 对象，然后使用 onDraw()方法绘制了不同填充样式的矩形、椭圆和多边形。主文件 example.java 的主要代码如下所示：

```
    /* 声明 Paint 对象 */
    private Paint mPaint = null;
    private example3_1 mGameView2 = null;
    public example(Context context)
    {
        super(context);
        /* 构建对象 */
        mPaint = new Paint();

        mGameView2 = new example3_1(context);

        /* 开启线程  */
        new Thread(this).start();
    }

    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);

        /* 设置画布为黑色背景 */
        canvas.drawColor(Color.BLACK);
        /* 取消锯齿 */
        mPaint.setAntiAlias(true);

        mPaint.setStyle(Paint.Style.STROKE);

        {
            /* 定义矩形对象 */
            Rect rect1 = new Rect();
            /* 设置矩形大小 */
            rect1.left = 5;
            rect1.top = 5;
            rect1.bottom = 25;
```

```
        rect1.right = 45;

        mPaint.setColor(Color.BLUE);
        /* 绘制矩形 */
        canvas.drawRect(rect1, mPaint);
        mPaint.setColor(Color.RED);
        /* 绘制矩形 */
        canvas.drawRect(50, 5, 90, 25, mPaint);
        mPaint.setColor(Color.YELLOW);
        /* 绘制圆形(圆心 x，圆心 y，半径 r,p) */
        canvas.drawCircle(40, 70, 30, mPaint);
        /* 定义椭圆对象 */
        RectF rectf1 = new RectF();
        /* 设置椭圆大小 */
        rectf1.left = 80;
        rectf1.top = 30;
        rectf1.right = 120;
        rectf1.bottom = 70;

        mPaint.setColor(Color.LTGRAY);
        /* 绘制椭圆 */
        canvas.drawOval(rectf1, mPaint);
        /* 绘制多边形 */
        Path path1 = new Path();
        /*设置多边形的点*/
        path1.moveTo(150+5, 80-50);
        path1.lineTo(150+45, 80-50);
        path1.lineTo(150+30, 120-50);
        path1.lineTo(150+20, 120-50);
        /* 使这些点构成封闭的多边形 */
        path1.close();
        mPaint.setColor(Color.GRAY);
        /* 绘制这个多边形 */
        canvas.drawPath(path1, mPaint);
        mPaint.setColor(Color.RED);
        mPaint.setStrokeWidth(3);
        /* 绘制直线 */
        canvas.drawLine(5, 110, 315, 110, mPaint);
    }
    //
    //下面绘制实心几何体
    //
    mPaint.setStyle(Paint.Style.FILL);
    {
        /* 定义矩形对象 */
        Rect rect1 = new Rect();
        /* 设置矩形大小 */
        rect1.left = 5;
        rect1.top = 130+5;
        rect1.bottom = 130+25;
```

```
        rect1.right = 45;
        mPaint.setColor(Color.BLUE);
        /* 绘制矩形 */
        canvas.drawRect(rect1, mPaint);
        mPaint.setColor(Color.RED);
        /* 绘制矩形 */
        canvas.drawRect(50, 130+5, 90, 130+25, mPaint);
        mPaint.setColor(Color.YELLOW);
        /* 绘制圆形(圆心 x，圆心 y，半径 r,p) */
        canvas.drawCircle(40, 130+70, 30, mPaint);
        /* 定义椭圆对象 */
        RectF rectf1 = new RectF();
        /* 设置椭圆大小 */
        rectf1.left = 80;
        rectf1.top = 130+30;
        rectf1.right = 120;
        rectf1.bottom = 130+70;
        mPaint.setColor(Color.LTGRAY);
        /* 绘制椭圆 */
        canvas.drawOval(rectf1, mPaint);
        /* 绘制多边形 */
        Path path1 = new Path();
        /*设置多边形的点*/
        path1.moveTo(150+5, 130+80-50);
        path1.lineTo(150+45, 130+80-50);
        path1.lineTo(150+30, 130+120-50);
        path1.lineTo(150+20, 130+120-50);
        /* 使这些点构成封闭的多边形 */
        path1.close();
        mPaint.setColor(Color.GRAY);
        /* 绘制这个多边形 */
        canvas.drawPath(path1, mPaint);

        mPaint.setColor(Color.RED);
        mPaint.setStrokeWidth(3);
        /* 绘制直线 */
        canvas.drawLine(5, 130+110, 315, 130+110, mPaint);
    }

    /* 通过 ShapeDrawable 来绘制几何图形 */
    mGameView2.DrawShape(canvas);
}
// 触笔事件
public boolean onTouchEvent(MotionEvent event)
{
    return true;
}
// 按键按下事件
```

```
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        return true;
    }
    // 按键弹起事件
    public boolean onKeyUp(int keyCode, KeyEvent event)
    {
        return false;
    }
    public boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event)
    {
        return true;
    }
    public void run()
    {
        while (!Thread.currentThread().isInterrupted())
        {
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
            //使用 postInvalidate 可以直接在线程中更新界面
            postInvalidate();
        }
    }
}
```

至此整个演练结束，执行后的效果如图 15-3 所示。

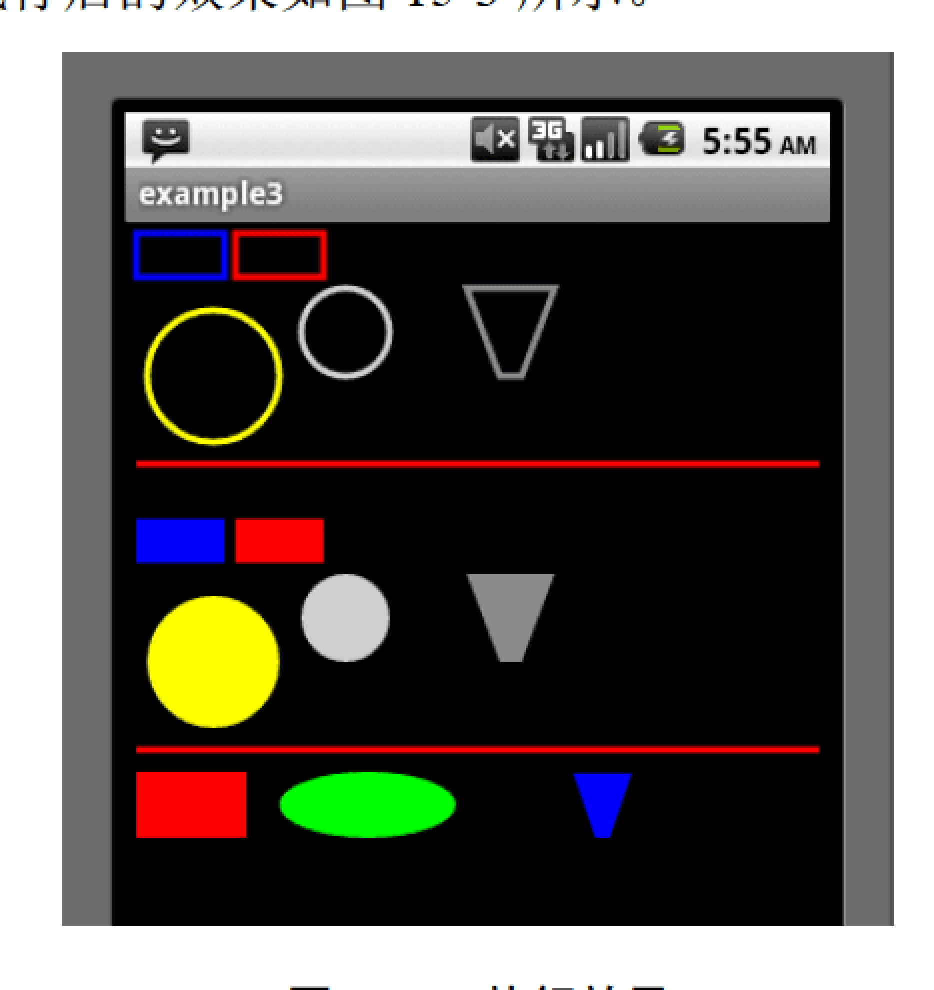


图 15-3　执行效果

15.1.5　NinePatch 类

NinePatch 类即 Android.Graphics.NinePatch，NinePatch 是 Android 平台特有的一种自然拉伸非矢量图形的方法，可以帮助常规的图形在拉伸时不会缩放，此类的最大作用是创建高质量的透明的可缩放图片，这种格式的图片非常适合在手机上使用。为了帮助程序员迅速找到合适的素材图片，在 Android 80k 中为我们提供了一个名为 Draw 9-patch 的工具，有关该工具的使用方法可参考相关资料，本书不再进行详细讲解。

15.1.6　Matrix 类

Matrix 类即 Android.Graphics.Matrix，能够实现图形的变换操作，例如常见的缩放和旋转处理。Matrix 中有关图形的变换、缩放等相关操作常用的方法有如下几种。

(1)　void reset()：重置一个 matrix 对象。

(2)　void set(Matrix src)：复制一个源矩阵，和本类的构造方法 Matrix(Matrix src)一样。

(3)　boolean isIdentity()：返回这个矩阵是否定义(已经有意义)。

(4)　void setRotate(float degrees)：指定一个角度以 0,0 为坐标进行旋转。

(5)　void setRotate(float degrees, float px, float py)：指定一个角度以 px,py 为坐标进行旋转。

(6)　void setScale(float sx, float sy)：缩放处理。

(7)　void setScale(float sx, float sy, float px, float py)：以坐标 px,py 进行缩放。

(8)　void setTranslate(float dx, float dy)：平移。

(9)　void setSkew (float kx, float ky, float px, float py)：以坐标 px, py 进行倾斜。

(10) void setSkew (float kx, float ky)：倾斜处理。

15.1.7　Bitmap 类

Bitmap 类即 Android.Graphics.Bitmap，是一个位图操作类，实现对位图的基本操作。Bitmap 中提供了很多实用的方法，其中最为常用的几种方法如下。

(1)　boolean compress(Bitmap.CompressFormat format, int quality, OutputStream stream)：压缩一个 Bitmap 对象，根据相关的编码、画质保存到一个 OutputStream 中，其中第一个压缩格式目前有 JPG 和 PNG。

(2)　void copyPixelsFromBuffer(Buffer src)：从一个 Buffer 缓冲区复制位图像素。

(3)　void copyPixelsToBuffer(Buffer dst)：将当前位图像素内容复制到一个 Buffer 缓冲区。

我们看到创建位图对象 createBitmap 包含了 6 种方法在目前的 Android 2.1 SDK 中，当然他们使用的是 API Level 均为 1，所以说从 Android 1.0 SDK 开始就支持了，所以大家可以放心使用。

(4)　下面的方法用于创建一个可以缩放的位图对象。代码如下：

```
static Bitmap createBitmap(Bitmap src)
static Bitmap createBitmap(int[] colors, int width, int height, Bitmap.Config
config)
static Bitmap createBitmap(int[] colors, int offset, int stride, int width, int
height, Bitmap.Config config)
```

```
static Bitmap createBitmap(Bitmap source, int x, int y, int width, int height,
Matrix m, boolean filter)
static Bitmap createBitmap(int width, int height, Bitmap.Config config)
static Bitmap createBitmap(Bitmap source, int x, int y, int width, int height)
static Bitmap createScaledBitmap(Bitmap src, int dstWidth, int dstHeight, boolean
filter)
```

(5) final int getHeight()：获取高度。

(6) final int getWidth()：获取宽度。

(7) final boolean hasAlpha()：是否有透明通道。

(8) void setPixel(int x, int y, int color)：设置某像素的颜色。

(9) int getPixel(int x, int y)：获取某像素的颜色。

题　目	目　的	源码路径
演练 4	使用 Bitmap 类实现模拟水纹效果	“光盘:\daima\15\example4”文件夹

在本实例中，使用类 Bitmap 装载了图片，并且定义方法 RippleSpread()分别实现了波能扩展和衰减结果，最终模拟实现了水纹效果。主文件 example4.java 的主要代码如下所示：

```
public class example4 extends View implements Runnable
{
    int BACKWIDTH;
    int BACKHEIGHT;
    short[] buf2;
    short[] buf1;
    int[] Bitmap2;
    int[] Bitmap1;
    public example4(Context context)
    {
        super(context);

        /* 装载图片 */
        Bitmap image = BitmapFactory.decodeResource(this.getResources(),
        R.drawable.qq);
        BACKWIDTH = image.getWidth();
        BACKHEIGHT = image.getHeight();

        buf2 = new short[BACKWIDTH * BACKHEIGHT];
        buf1 = new short[BACKWIDTH * BACKHEIGHT];
        Bitmap2 = new int[BACKWIDTH * BACKHEIGHT];
        Bitmap1 = new int[BACKWIDTH * BACKHEIGHT];
        /* 加载图片的像素到数组中 */
        image.getPixels(Bitmap1, 0, BACKWIDTH, 0, 0, BACKWIDTH, BACKHEIGHT);

        new Thread(this).start();
    }


    void DropStone(int x,// x 坐标
                   int y,// y 坐标
```

```
                int stonesize,// 波源半径
                int stoneweight)// 波源能量
{
    for (int posx = x - stonesize; posx < x + stonesize; posx++)
        for (int posy = y - stonesize; posy < y + stonesize; posy++)
            if ((posx - x) * (posx - x) + (posy - y) * (posy - y) < stonesize
            * stonesize)
                buf1[BACKWIDTH * posy + posx] = (short) -stoneweight;
}


void RippleSpread()
{
    for (int i = BACKWIDTH; i < BACKWIDTH * BACKHEIGHT - BACKWIDTH; i++)
    {
        // 波能扩散
        buf2[i] = (short) (((buf1[i - 1] + buf1[i + 1] + buf1[i - BACKWIDTH]
        + buf1[i + BACKWIDTH]) >> 1) - buf2[i]);
        // 波能衰减
        buf2[i] -= buf2[i] >> 5;
    }

    // 交换波能数据缓冲区
    short[] ptmp = buf1;
    buf1 = buf2;
    buf2 = ptmp;
}

/* 渲染水纹效果 */
void render()
{
    int xoff, yoff;
    int k = BACKWIDTH;
    for (int i = 1; i < BACKHEIGHT - 1; i++)
    {
        for (int j = 0; j < BACKWIDTH; j++)
        {
            // 计算偏移量
            xoff = buf1[k - 1] - buf1[k + 1];
            yoff = buf1[k - BACKWIDTH] - buf1[k + BACKWIDTH];

            // 判断坐标是否在窗口范围内
            if ((i + yoff) < 0)
            {
                k++;
                continue;
            }
            if ((i + yoff) > BACKHEIGHT)
            {
                k++;
```

```
                        continue;
                    }
                    if ((j + xoff) < 0)
                    {
                        k++;
                        continue;
                    }
                    if ((j + xoff) > BACKWIDTH)
                    {
                        k++;
                        continue;
                    }

                    // 计算出偏移像素和原始像素的内存地址偏移量
                    int pos1, pos2;
                    pos1 = BACKWIDTH * (i + yoff) + (j + xoff);
                    pos2 = BACKWIDTH * i + j;
                    Bitmap2[pos2++] = Bitmap1[pos1++];
                    k++;
                }
            }
        }

        public void onDraw(Canvas canvas)
        {
            super.onDraw(canvas);

            /* 绘制经过处理的图片效果 */
            canvas.drawBitmap(Bitmap2, 0, BACKWIDTH, 0, 0, BACKWIDTH, BACKHEIGHT,
            false, null);
        }

        // 触笔事件
        public boolean onTouchEvent(MotionEvent event)
        {

            return true;
        }

        // 按键按下事件
        public boolean onKeyDown(int keyCode, KeyEvent event)
        {
            return true;
        }

        // 按键弹起事件
        public boolean onKeyUp(int keyCode, KeyEvent event)
        {
            DropStone(BACKWIDTH/2, BACKHEIGHT/2, 10, 30);
            return false;
```

```
    }


    public boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event)
    {
        return true;
    }

    /**
     * 线程处理
     */
    public void run()
    {
        while (!Thread.currentThread().isInterrupted())
        {
            try
            {
                Thread.sleep(50);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
           RippleSpread();
           render();
            //使用 postInvalidate 可以直接在线程中更新界面
            postInvalidate();
        }
    }
}
```

至此整个演练结束，执行后将通过对图像像素的操作数来模拟水纹效果，执行效果如图 15-4 所示。

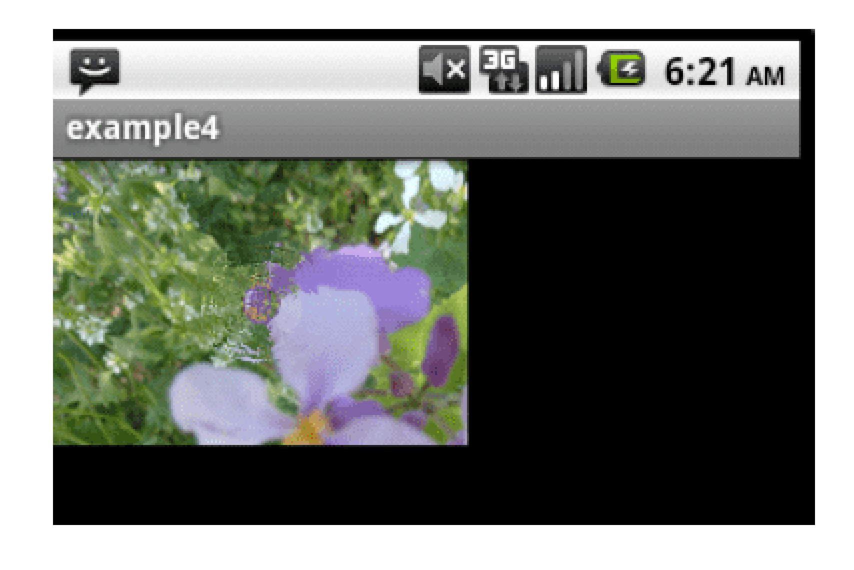


图 15-4　执行效果

15.1.8　BitmapFactory 类

BitmapFactory 类即 Android.Graphics.BitmapFactory，作为 Bitmap 对象的 I/O 类，BitmapFactory 类提供了丰富的构造 Bitmap 对象的方法，比如从一个字节数组、文件系统、资源 ID 以及输入流中来创建一个 Bitmap 对象，该类的全部成员中除了 decodeFileDescriptor 外其他的重载方法都

很常用。

(1) 从字节数组创建，代码如下：

```
static Bitmap decodeByteArray(byte[] data, int offset, int length)
static Bitmap decodeByteArray(byte[] data, int offset, int length, BitmapFactory.
Options opts)
```

(2) 从文件创建，路径要写全，代码如下：

```
static Bitmap decodeFile(String pathName, BitmapFactory.Options opts)
static Bitmap decodeFile(String pathName)
```

(3) 从输入流句柄创建，代码如下：

```
static Bitmap decodeFileDescriptor(FileDescriptor fd, Rect outPadding, BitmapFactory.
Options opts)
static Bitmap decodeFileDescriptor(FileDescriptor fd)
```

(4) 从 Android 的 APK 文件资源中创建，代码如下：

```
static Bitmap decodeResource(Resources res, int id)
static Bitmap decodeResource(Resources res, int id, BitmapFactory.Options opts)
static Bitmap decodeResourceStream(Resources res, TypedValue value, InputStream
is, Rect pad, BitmapFactory.Options opts)
```

(5) 从一个输入流中创建，代码如下：

```
static Bitmap decodeStream(InputStream is)
static Bitmap decodeStream(InputStream is, Rect outPadding, BitmapFactory.
Options opts)
```

15.1.9 Region 类

Region 类即 Android.Graphics.Region，Region 在 Android 平台中表示一个区域，和 Rect 类不同的是，它表示的是一个不规则的样子，可以是椭圆、多边形等，而 Rect 仅仅是矩形。同样 Region 的 boolean contains(int x, int y)成员可以判断一个点是否在该区域内。

15.1.10 Typeface 类

Typeface 类即 Android.Graphics.Typeface，Typeface 类是帮助描述一个字体对象，在 TextView 中通过使用 setTypeface 方法来制定一个输出文本的字体，其直接构造调用成员 create 方法可以指定一个字体名称和样式，例如下面的代码：

```
static Typeface create(Typeface family, int style)
static Typeface create(String familyName, int style)
```

同时使用 isBold 和 isItalic 方法可以判断出是否包含粗体或斜体的字型：

```
final boolean  isBold()
final boolean  isItalic()
```

该类的创建方法还有从 APK 的资源或从一个具体的文件路径来创建，其代码如下：

```
static Typeface  createFromAsset(AssetManager mgr, String path)
```

```
static Typeface  createFromFile(File path)
static Typeface  createFromFile(String path)
```

15.1.11　Shader 类

题　目	目　的	源码路径
演练 5	用 Shader 类来渲染不同的图像	“光盘:\daima\15\example5”文件夹

在本实例中，首先声明了 Bitmap 对象，并且分别实现了线性渐变渲染、混合渲染、唤醒渐变渲染和梯度渲染。然后在定义的类 example5 中，使用 Shader 类渲染了不同的图像。

主文件 example5.java 的具体代码如下所示：

```
/* 声明 Bitmap 对象 */
Bitmap  mBitQQ  = null;
int     BitQQwidth  = 0;
int     BitQQheight = 0;
Paint   mPaint = null;

/* Bitmap 渲染 */
Shader mBitmapShader = null;
/* 线性渐变渲染 */
Shader mLinearGradient = null;
/* 混合渲染 */
Shader mComposeShader = null;
/* 唤醒渐变渲染 */
Shader mRadialGradient = null;
/* 梯度渲染 */
Shader mSweepGradient = null;
ShapeDrawable mShapeDrawableQQ = null;
public example5(Context context)
{
    super(context);

    /* 装载资源 */
    mBitQQ = ((BitmapDrawable) getResources().getDrawable(R.drawable.qq)).
    getBitmap();

    /* 得到图片的宽度和高度 */
    BitQQwidth  = mBitQQ.getWidth();
    BitQQheight = mBitQQ.getHeight();

    /* 创建 BitmapShader 对象 */
    mBitmapShader = new BitmapShader(mBitQQ,Shader.TileMode.REPEAT,Shader.
    TileMode.MIRROR);

    /* 创建 LinearGradient 并设置渐变的颜色数组 */
    mLinearGradient = new LinearGradient(0,0,100,100, new
                          int[]{Color.RED,Color.GREEN,Color.BLUE,Color.WHITE},
```

```
                        null,Shader.TileMode.REPEAT);
        /* 这里笔者理解为“混合渲染”--大家可以有自己的理解，能明白这个意思就好*/
        mComposeShader = new ComposeShader(mBitmapShader,mLinearGradient,
        PorterDuff.Mode.DARKEN);

        /* 构建 RadialGradient 对象，设置半径的属性 */
        //这里使用了 BitmapShader 和 LinearGradient 进行混合
        //当然也可以使用其他的组合
        //混合渲染的模式很多，可以根据自己的需要来选择
        mRadialGradient = new RadialGradient(50,200,50, new int[]{Color.GREEN,
        Color.RED,Color.BLUE,Color.WHITE},null,Shader.TileMode.REPEAT);
        /* 构建 SweepGradient 对象 */
       mSweepGradient = new SweepGradient(30,30,new int[]{Color.GREEN,Color.RED,
        Color.BLUE,Color.WHITE},null);
        mPaint = new Paint();

        /* 开启线程 */
        new Thread(this).start();
    }

    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);

        //将图片裁剪为椭圆形
        /* 构建 ShapeDrawable 对象并定义形状为椭圆 */
        mShapeDrawableQQ = new ShapeDrawable(new OvalShape());

        /* 设置要绘制的椭圆形的东西为 ShapeDrawable 图片 */
        mShapeDrawableQQ.getPaint().setShader(mBitmapShader);

        /* 设置显示区域 */
        mShapeDrawableQQ.setBounds(0,0, BitQQwidth, BitQQheight);

        /* 绘制 ShapeDrawableQQ */
        mShapeDrawableQQ.draw(canvas);

        //绘制渐变的矩形
        mPaint.setShader(mLinearGradient);
        canvas.drawRect(BitQQwidth, 0, 320, 156, mPaint);

        //显示混合渲染效果
        mPaint.setShader(mComposeShader);
        canvas.drawRect(0, 300, BitQQwidth, 300+BitQQheight, mPaint);

        //绘制环形渐变
        mPaint.setShader(mRadialGradient);
        canvas.drawCircle(50, 200, 50, mPaint);
```

```
        //绘制梯度渐变
        mPaint.setShader(mSweepGradient);
        canvas.drawRect(150, 160, 300, 300, mPaint);
    }

    // 触笔事件
    public boolean onTouchEvent(MotionEvent event)
    {
        return true;
    }
    // 按键按下事件
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        return true;
    }
    // 按键弹起事件
    public boolean onKeyUp(int keyCode, KeyEvent event)
    {
        return false;
    }
    public boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event)
    {
        return true;
    }
    /** 线程处理 */
    public void run()
    {
        while (!Thread.currentThread().isInterrupted())
        {
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
            //使用 postInvalidate 可以直接在线程中更新界面
            postInvalidate();
        }
    }
}
```

至此整个演练结束，执行后的效果如图 15-5 所示。

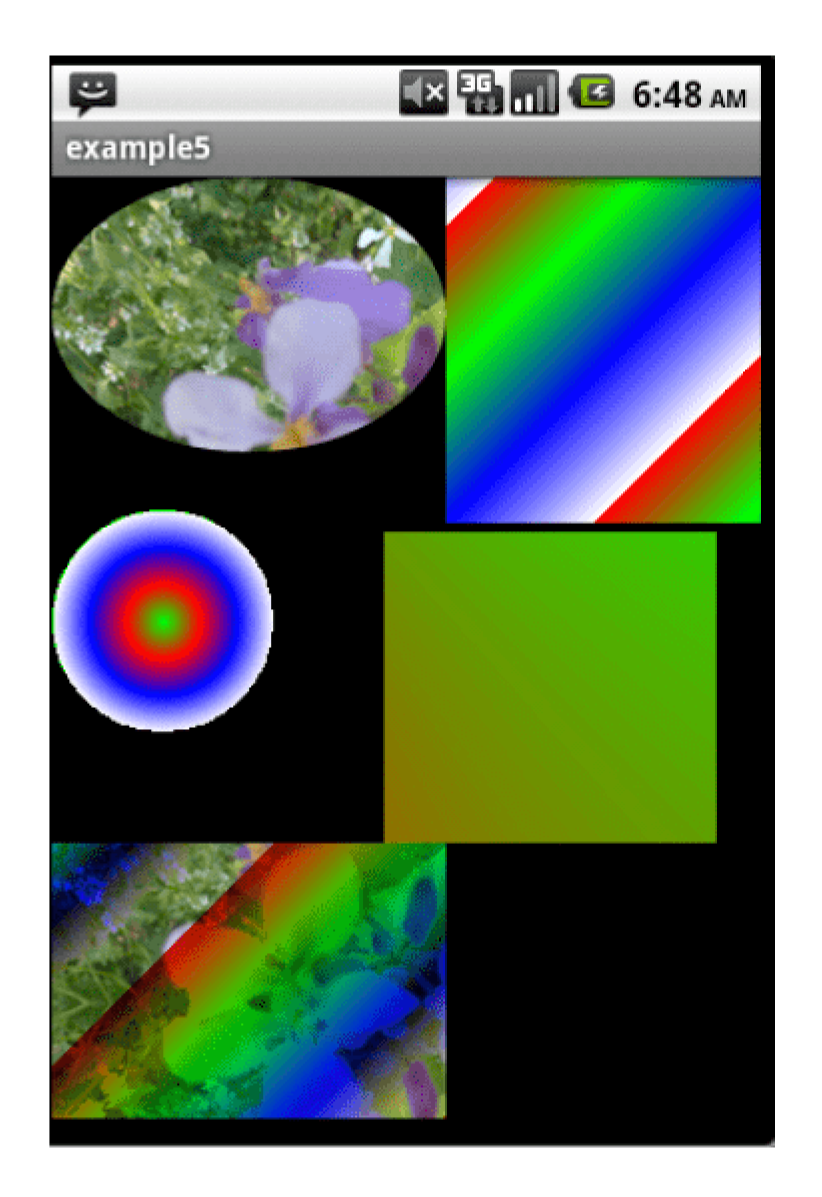


图 15-5　执行效果

15.2　动画美轮美奂

安卓心法博大精深，美轮美奂的动画竟然也能够实现。在 Android 平台中提供了两类动画，分别是 Tween 动画和 Frame 动画。Tween 动画用于对场景里的对象不断地进行图像变换来产生动画效果；Frame 动画用于顺序播放事先做好的图像。

15.2.1　Tween 动画

题　目	目　的	源码路径
演练 6	在 Android 中使用 Tween 动画	“光盘:\daima\15\example9”文件夹

在本实例中分别定义了 Alpha 动画、Scale 动画、Rotate 动画，然后在定义的类 example6 中实现了 Tween 动画效果。主文件 example9.java 的主要代码如下所示：

```
/* 定义 Alpha 动画 */
private Animation mAnimationAlpha = null;

/* 定义 Scale 动画 */
private Animation  mAnimationScale = null;

/* 定义 Translate 动画 */
private Animation mAnimationTranslate = null;

/* 定义 Rotate 动画 */
private Animation mAnimationRotate = null;

/* 定义 Bitmap 对象 */
Bitmap mBitQQ = null;
```

```
public example6(Context context)
{
    super(context);

    /* 装载资源 */
    mBitQQ = ((BitmapDrawable) getResources().getDrawable
    (R.drawable.qq)).getBitmap();
}

public void onDraw(Canvas canvas)
{
    super.onDraw(canvas);

    /* 绘制图片 */
    canvas.drawBitmap(mBitQQ, 0, 0, null);
}

public boolean onKeyUp(int keyCode, KeyEvent event)
{
    switch ( keyCode )
    {
    case KeyEvent.KEYCODE_DPAD_UP:
        /* 创建 Alpha 动画 */
        mAnimationAlpha = new AlphaAnimation(0.1f, 1.0f);
        /* 设置动画的时间 */
        mAnimationAlpha.setDuration(3000);
        /* 开始播放动画 */
        this.startAnimation(mAnimationAlpha);
        break;
    case KeyEvent.KEYCODE_DPAD_DOWN:
        /* 创建 Scale 动画 */
        mAnimationScale =new ScaleAnimation(0.0f, 1.0f, 0.0f, 1.0f,
        Animation.RELATIVE_TO_SELF, 0.5f,
        Animation.RELATIVE_TO_SELF, 0.5f);
        /* 设置动画的时间 */
        mAnimationScale.setDuration(500);
        /* 开始播放动画 */
        this.startAnimation(mAnimationScale);
        break;
    case KeyEvent.KEYCODE_DPAD_LEFT:
        /* 创建 Translate 动画 */
        mAnimationTranslate = new TranslateAnimation(10, 100,10, 100);
        /* 设置动画的时间 */
        mAnimationTranslate.setDuration(1000);
        /* 开始播放动画 */
        this.startAnimation(mAnimationTranslate);
        break;
    case KeyEvent.KEYCODE_DPAD_RIGHT:
        /* 创建 Rotate 动画 */
        mAnimationRotate=new RotateAnimation(0.0f, +360.0f,
        Animation.RELATIVE_TO_SELF,0.5f,
```

```
                Animation.RELATIVE TO SELF, 0.5f);
                /* 设置动画的时间 */
                mAnimationRotate.setDuration(1000);
                /* 开始播放动画 */
                this.startAnimation(mAnimationRotate);
                break;
        }
        return true;
    }
}
```

程序执行后，将会把指定的目标图片模拟为动画显示，执行效果如图 15-6 所示。

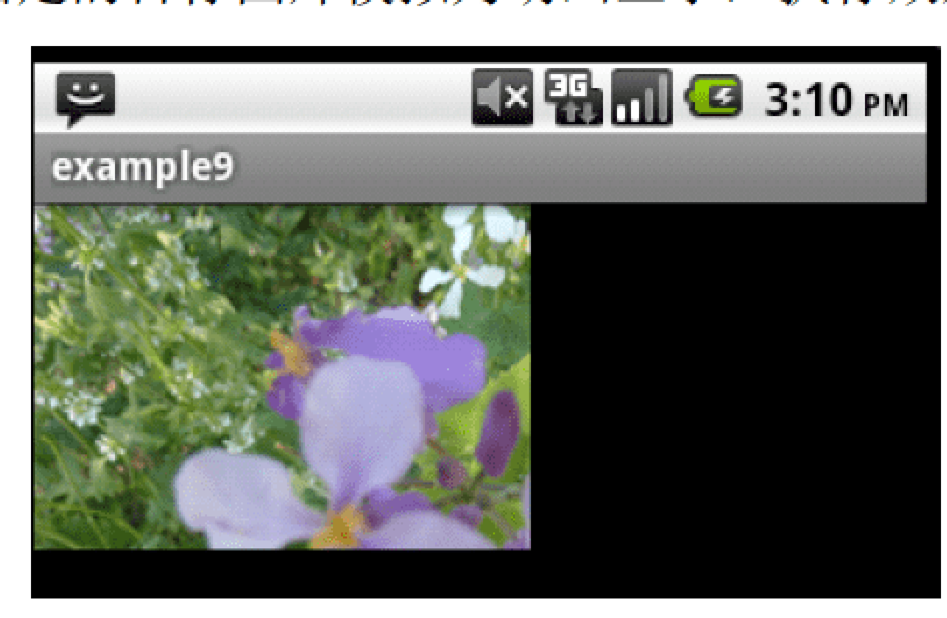


图 15-6　执行效果

15.2.2　Frame 动画

题　目	目　的	源码路径
演练 7	在 Android 中使用 Frame 动画	“光盘:\daima\15\example10”文件夹

在本实例中，首先定义了一个 DraWable 对象，然后装载了一幅图片资源并通过 Frame 播放了预设的资源，最终实现 Frame 动画效果。主文件 example10.java 的主要代码如下所示：

```
    /* 定义 AnimationDrawable 动画 */
    private AnimationDrawable  frameAnimation  = null;
    Context mContext = null;
    /* 定义一个 Drawable 对象 */
    Drawable mBitAnimation = null;
    public example10(Context context)
    {
        super(context);
        mContext = context;
        /* 实例化 AnimationDrawable 对象 */
        frameAnimation = new AnimationDrawable();
        /* 装载资源 */
        //这里用一个循环装载所有名字类似的资源
        //如“a1.......15.png”的图片
        //这个方法用处非常大
        for (int i = 1; i <= 15; i++)
        {
```

```
            int id = getResources().getIdentifier("a" + i, "drawable", mContext.
            getPackageName());
            mBitAnimation = getResources().getDrawable(id);
            /* 为动画添加一帧 */
            //参数 mBitAnimation 是该帧的图片
            //参数 500 是该帧显示的时间，按毫秒计算
            frameAnimation.addFrame(mBitAnimation, 500);
        }

        /* 设置播放模式是否循环。false 表示循环，而 true 表示不循环 */
        frameAnimation.setOneShot( false );

        /* 设置本类将要显示这个动画 */
        this.setBackgroundDrawable(frameAnimation);
    }

    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
    }
    public boolean onKeyUp(int keyCode, KeyEvent event)
    {
        switch ( keyCode )
        {
        case KeyEvent.KEYCODE_DPAD_UP:
            /* 开始播放动画 */
            frameAnimation.start();
            break;
        }
        return true;
    }
}
```

实例执行后，通过按下键盘的上、下方向键，能够实现动画效果，如图 15-7 所示。



图 15-7　执行效果

第 16 章　虚拟与现实并不远

手机游戏已经成为当前的主流游戏产业。随着技术的不断创新和硬件的提高，手机 3D 游戏逐渐进入到普通用户生活。移动价值链上所有环节的领先厂商也进一步加深彼此之间的合作，3D 游戏必将获得更快的发展。安卓中提供了 android.OpenGL 包，专门用于 3D 的加速和渲染处理。在本章的内容中，将详细讲解 OpenGL 的基本知识，为读者步入手机游戏开发领域打下基础。

16.1　OpenGL

OpenGL 的前身是 SGI 公司为其图形工作站开发的 IRIS GL。IRIS GL 是一个工业标准的 3D 图形软件接口，功能虽然强大但是移植性不好，于是 SGI 公司便在 IRIS GL 的基础上开发了 OpenGL。OpenGL 的英文全称是 Open Graphics Library，顾名思义，OpenGL 便是“开放的图形程序接口”。虽然 DirectX 在家用市场全面领先，但在专业高端绘图领域，OpenGL 是不能被取代的。

16.1.1　OpenGL 的发展历程

1992 年 7 月，SGI 公司发布了 OpenGL 1.0 版本，随后又与微软公司共同开发了 Windows NT 版本的 OpenGL，从而使一些原来必须在高档图形工作站上运行的大型 3D 图形处理软件也可以在微机上运用。

1995 年 OpenGL 1.1 版本面市，该版本较 1.0 性能提高许多，并加入了一些新功能，包括提高顶点位置、法线、颜色、色彩指数、纹理坐标、多边形边缘标识的传输速度，引入了新的纹理特性等。

1997 年，Windows 95 下 3D 游戏的大量涌现，游戏开发公司迫切需要一个功能强大、兼容性好的 3D 图形接口，而当时微软公司自己的 3D 图形接口 DirectX 3.0 功能却很糟糕。因而以制作《雷神之锤》等经典 3D 射击游戏而著名的 id 公司同其他一些游戏开发公司一同强烈要求微软在 Windows 95 中加入对 OpenGL 的支持。微软公司最终在 Windows 95 的 OSR2 版和后来的 Windows 版本中加入了对 OpenGL 的支持。这样，不但许多支持 OpenGL 的电脑 3D 游戏得到广泛应用，而且许多在 3D 图形设计软件也可以运用支持 OpenGL 标准的 3D 加速卡，大大提高

3D 图形的处理速度。

2003 年 7 月 28 日，SGI 和 ARB 公布了 OpenGL 1.5 版本。OpenGL 1.5 中包括 OpenGL ARB 的正式扩展规格绘制语言 OpenGL Shading Language。OpenGL 1.5 的新功能包括顶点 Buffer Object、Shadow 功能、隐蔽查询、非乘方纹理等。

2004 年 8 月，OpenGL 2.0 版本发布。OpenGL 2.0 标准的主要制订者并非原来的 SGI，而是逐渐在 ARB 中占据主动地位的 3Dlabs。OpenGL 2.0 支持 OpenGL Shading Language，新的 Shader 扩展特性以及其他多项增强特性。

2008 年 8 月，Khronos 工作组在 Siggraph 2008 大会上宣布了 OpenGL 3.0 图形接口规范，GLSL1.30 Shader 语言和其他新增功能将再次为未来开放 3D 接口发展指明了方向。

OpenGL 3.0 API 开发代号为 Longs Peak，和以往一样，OpenGL 3.0 仍然作为一个开放性和跨平台的 3D 图形接口标准，在 Shader 语言盛行的今天，OpenGL3.0 增加了新版本的 Shader 语言：GLSL 1.30 可以充分发挥当前可编程图形硬件的潜能。同时，OpenGL3.0 还引入了一些新的功能，例如：顶点矩阵对象、全帧缓存对象功能、32bit 浮点纹理和渲染缓存、基于阻塞队列的条件渲染、紧凑行半浮点顶点和像素数据，四个新压缩机制等。

2009 年 3 月又公布了升级版新规范 OpenGL 3.1。OpenGL 3.1 将此前引入的 OpenGL 着色语言 GLSL 从 1.30 版升级到了 1.40 版，通过改进程序增强了对最新可编程图形硬件的访问，还有更高效的顶点处理、扩展的纹理功能、更弹性的缓冲管理等。宽泛地讲，OpenGL 3.1 在 3.0 版的基础上对整个 API 模型体系进行了简化，可大幅提高软件开发效率。

2009 年 8 月 Khronos 工作组发布了 OpenGL 3.2 级。该版本仍然延续了 OpenGL 发展的方向让图形程序开发者能在多种操作系统和平台下更好地利用新的 GPU 功能。OpenGL3.2 版本提升了性能表现、改进了视觉质量、提高了几何图形处理速度，而且使 Direct3D 程序更容易移植为 OpenGL。除 OpenGL 之外，Khronos 还将其开发的其他标准进行了协调改进，以求可以在更广泛的领域提供强大的图形功能和计算生态系统，这些标准包括用于并行计算的 OpenCL、用于移动 3D 图形开发的 OpenGL ES 和用于网络 3D 开发的 WebGL。

2010 年 7 月 26 日发布了 OpenGL 4.1 和 OpenGL OpenGL Shading Language 4.10 版本。OpenGL 4.1 提高了视觉密集型应用 OpenCL™的互操作性，并继续加速计算剖面为核心的支持和兼容 OpenGL 3.2，使开发人员能够使用一个简化的 API 或保留向后兼容现有的 OpenGL 代码，这取决于他们的市场需求。

16.1.2　OpenGL 的特点和功能

OpenGL 是一个开放的三维图形软件包，它独立于窗口系统和操作系统，以它为基础开发的应用程序可以十分方便地在各种平台间移植。OpenGL 可以与 Visual C++紧密接口，便于实现机械手的有关计算和图形算法，可保证算法的正确性和可靠性。OpenGL 使用简便，效率高，它具有如下 7 大功能。

(1)　建模：OpenGL 图形库除了提供基本的点、线、多边形的绘制函数外，还提供了复杂的三维物体(球、锥、多面体、茶壶等)以及复杂曲线和曲面绘制函数。

(2)　变换：OpenGL 图形库的变换包括基本变换和投影变换。基本变换有平移、旋转、变比镜像四种变换，投影变换有平行投影(又称正射投影)和透视投影两种变换。其变换方法有利于减少算法的运行时间，提高三维图形的显示速度。

(3)　颜色模式设置：OpenGL 颜色模式有两种，即 RGBA 模式和颜色索引(Color Index)。

(4)　光照和材质设置：OpenGL 光有辐射光(Emitted Light)、环境光(Ambient Light)、漫反射光(Diffuse Light)和镜面光(Specular Light)。材质是用光反射率来表示的，场景(Scene)中物体最终反映到人眼的颜色是光的红绿蓝分量与材质红绿蓝分量的反射率相乘后形成的颜色。

(5)　纹理映射(Texture Mapping)：利用 OpenGL 纹理映射功能可以十分逼真地表达物体表面的细节。

(6)　位图显示和图像增强图象功能除了基本的拷贝和像素读写外，还提供融合(Blending)、反走样(Antialiasing)和雾(fog)的特殊图像效果处理。可使被仿真物更具真实感，增强图形显示的效果。

(7)　双缓存动画(Double Buffering)：双缓存即前台缓存和后台缓存，简言之，后台缓存计算场景、生成画面，前台缓存显示后台缓存已画好的画面。

16.1.3　为移动设备而生的 OpenGL ES

OpenGL ES (OpenGL for Embedded Systems) 是 OpenGL 三维图形 API 的子集，针对手机、PDA 和游戏主机等嵌入式设备而设计。该 API 由 Khronos 组织定义推广，Khronos 是一个图形软硬件行业协会，该协会主要关注图形和多媒体方面的开放标准。

Android 系统使用 OpenGL 的标准接口来支持 3D 图形功能，Android 3D 图形系统也分为 Java 框架和本地代码两部分。本地代码主要实现 OpenGL 接口库，在 Java 框架层，javax.microedition.khronos.opengles 是 Java 标准的 OpenGL 包，android.opengl 包提供了 OpenGL 系统和 Android GUI 系统之间的联系。

Android 的本地代码位于如下目录：

frameworks/base/opengl

JNI 代码位于如下目录：

frameworks/base/core/com_google_android_gles_jni_GLImpl.cpp，

或如下目录：

frameworks/base/core/com_google_android_gles_jni_EGLImpl.cpp

Java 类位于如下目录：

opengl/java/javax/microedition/khronos

16.1.4　Android OpenGL ES 密录

1. OpenGL ES 测试代码

在 frameworks/base/opengl/tests 目录下有 OpenGL 的本地测试代码，包括 angeles、fillrate 等 14 个测试代码，这些代码都可以通过终端进行本地调用测试(模拟器中使用 adb shell)。在 tests 文件夹中执行 mm，将输出以下信息：

```
Install: out/target/product/generic/system/bin/angeles
…
Install: out/target/product/generic/system/bin/test-opengl-tritex
```

由以上信息可知，测试用例被安装在了 out/target/product/generic/system/bin/目录下，将其拷

贝到 nfs 文件系统中便于测试，把这些测试用例单独放在 Android 的根文件系统的 gltest 文件夹中。

2. OpenGL ES 的绘图流程

使用 OpenGL ES 进行绘图的基本流程如下。

(1) 获取 Display，Display 代表显示器。函数原型如下：

```
EGLDisplay eglGetDisplay(NativeDisplayType display);
```

其中 display 参数是 native 系统的窗口显示 ID 值，一般为 EGL_DEFAULT_DISPLAY。该参数的实际意义是与平台实现相关，在 X-Window 下是 XDisplay ID，在 MS Windows 下是 Window DC。

(2) 初始化 egl 库，函数原型如下：

```
EGLBoolean eglInitialize(EGLDisplay dpy, EGLint *major, EGLint *minor);
```

其中 dpy 是一个有效的 EGLDisplay，函数返回时，major 和 minor 将被赋予当前 EGL 的版本号。

(3) 选择一个合适的 EGL Configuration FrameBuffer，实际指的是 FrameBuffer 的参数，函数原型如下：

```
EGLBoolean eglChooseConfig(EGLDisplay dpy, const EGLint *attrib_list,
EGLConfig *configs, EGLint config_size, EGLint *num_config);
```

各个参数的具体说明如下。

- 参数 attrib_list：指定了选择配置时需要参照的属性。
- 参数 configs：将返回一个按照 attrib_list 排序的平台所有有效的 EGL framebuffer 配置列表。
- 参数 config_size：指定了可以返回到 configs 的总配置个数。
- 参数 num_config：返回了实际匹配的配置总数。

(4) 创建一个可实际显示的 EGLSurface，实际上就是一个 FrameBuffer，函数原型如下：

```
EGLSurface eglCreateWindowSurface(EGLDisplay dpy, EGLConfig config,
NativeWindowType win, const EGLint *attrib_list);
```

(5) 创建 Context，函数原型如下：

```
EGLContext eglCreateContext(EGLDisplay dpy, EGLConfig config,
EGLContext share_context, const EGLint *attrib_list);
```

(6) 绑定 Display、Surface、Context，函数原型如下：

```
EGLBoolean eglMakeCurrent(EGLDisplay dpy, EGLSurface draw,
EGLSurface read, EGLContext ctx);
```

16.2 实战应用 Android OpenGL

对 Android OpenGL 我有了一个清晰的认识，看似简单实则功能强大！为了巩固所学的知识，通过下面几个实例，帮助读者加深理解。

16.2.1　移动的图像

题　目	目　的	源码路径
题目 1	实现移动星星的效果	“光盘:\daima\16\example_13_01”文件夹

这个题目很简单，需要事先准备一章素材图像，然后在此图像的基础上形成一个螺旋图案，以达到闪烁星星的效果。为了实现旋转，我定义了变量 angle 来表示当前星星所处的角度，下面是具体的实现流程。

(1)　在布局文件中设置插入一个 TextView。

(2)　编写主程序文件 GLRender.java，用于创建循环设置所有的星星并绘制每一个闪烁的星星。具体实现代码如下所示：

```
public class GLRender implements Renderer
{
    public static final int num = 50;           // 星星数目
    boolean twinkle = true;                     // 闪烁的星星
    boolean key;
    public Star[] star = new Star[num];         // 存放星星的数组
    float zoom = -10.0f;                        // 星星离观察者的距离
    float tilt = 90.0f;                         // 星星的倾角
    float spin;                                 // 闪烁星星的自转
    int one = 0x10000;
    Random random = new Random();
    Int texture;                                // 纹理

    IntBuffer coord = IntBuffer.wrap(new int[]{
            0,0,
            one,0,
            one,one,
            0,one,
    });
    IntBuffer vertexs = IntBuffer.wrap(new int[]{
            -one,-one,0,
            one,-one,0,
            one,one,0,
            -one,one,0,
    });
    ByteBuffer indices = ByteBuffer.wrap(new byte[]{
            1, 0, 2, 3
    });

    @Override
    public void onDrawFrame(GL10 gl)
    {

        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
          // 清除屏幕和深度缓存
        gl.glBindTexture(GL10.GL_TEXTURE_2D, texture); // 选择纹理

        for (int i=0; i<num; i++)                      // 循环设置所有的星星
```

```
    {
        gl.glLoadIdentity();            // 绘制每颗星星之前，重置模型观察矩阵
        gl.glTranslatef(0.0f,0.0f,zoom);           // 深入屏幕里面
        gl.glRotatef(tilt,1.0f,0.0f,0.0f);         // 倾斜视角
        gl.glRotatef(star[i].angle,0.0f,1.0f,0.0f);// 旋转至当前所画星星的角度
        gl.glTranslatef(star[i].dist,0.0f,0.0f);   // 沿 x 轴正向移动
        gl.glRotatef(-star[i].angle,0.0f,1.0f,0.0f);   // 取消当前星星的角度
        gl.glRotatef(-tilt,1.0f,0.0f,0.0f);            // 取消屏幕倾斜

        //设置定点数组
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        //设置颜色数组
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);

        gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);

        if (twinkle)                                   // 启用闪烁效果
        {
            // 使用 byte 型数值指定一个颜色

gl.glColor4f((float)star[(num-i)-1].r/255.0f,(float)star[(num-i)-1].g/25
 5.0f,(float)star[(num-i)-1].b/255.0f,1.0f);
            gl.glVertexPointer(3, GL10.GL_FIXED, 0, vertexs);
            gl.glTexCoordPointer(2, GL10.GL_FIXED, 0, coord);

            {
                coord.position(0);
                vertexs.position(0);
                indices.position(0);

                gl.glDrawElements(GL10.GL_TRIANGLE_STRIP, 4, GL10.GL_
                  UNSIGNED_BYTE, indices);
            }
            //绘制结束
            gl.glFinish();
        }

        gl.glRotatef(spin,0.0f,0.0f,1.0f);             // 绕 z 轴旋转星星

        // 使用 byte 型数值指定一个颜色

gl.glColor4f((float)star[(num-i)-1].r/255.0f,(float)star[(num-i)-1].g/255.0f,
  (float)star[(num-i)-1].b/255.0f,1.0f);
        gl.glVertexPointer(3, GL10.GL_FIXED, 0, vertexs);
        gl.glTexCoordPointer(2, GL10.GL_FIXED, 0, coord);

        {
            coord.position(0);
            vertexs.position(0);
            indices.position(0);
```

```
            gl.glDrawElements(GL10.GL_TRIANGLE_STRIP, 4, GL10.GL_UNSIGNED_
              BYTE, indices);
        }
        //绘制正方形结束
        gl.glFinish();

        gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
        //取消顶点数组
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDisableClientState(GL10.GL TEXTURE COORD ARRAY);

        spin+=0.01f;                                // 星星的公转
        star[i].angle+=(float)(i)/(float)num;       // 改变星星的自转角度
        star[i].dist-=0.01f;                        // 改变星星离中心的距离

        if (star[i].dist<0.0f)                      // 星星到达中心了么
        {
            star[i].dist+=5.0f;                     // 往外移 5 个单位
            star[i].r=random.nextInt(256);          // 赋一个新红色分量
            star[i].g=random.nextInt(256);          // 赋一个新绿色分量
            star[i].b=random.nextInt(256);          // 赋一个新蓝色分量
        }
    }
}
@Override
public void onSurfaceChanged(GL10 gl, int width, int height)
{
    float ratio = (float) width / height;
    //设置 OpenGL 场景的大小
    gl.glViewport(0, 0, width, height);
    //设置投影矩阵
    gl.glMatrixMode(GL10.GL_PROJECTION);
    //重置投影矩阵
    gl.glLoadIdentity();
    // 设置视口的大小
    gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
    // 选择模型观察矩阵
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    // 重置模型观察矩阵
    gl.glLoadIdentity();
}

@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
    gl.glShadeModel(GL10.GL_SMOOTH);                        // 启用阴影平滑
    gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);                // 黑色背景
    gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);
    // 告诉系统对透视进行修正
```

```
        IntBuffer buffer = IntBuffer.allocate(1);
        // 创建一个纹理
        gl.glGenTextures(1, buffer);
        texture = buffer.get();
        // 创建一个线性滤波纹理
        gl.glBindTexture(GL10.GL_TEXTURE_2D, texture);
        gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_
          LINEAR);
        gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_
          LINEAR);
        GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, GLImage.mBitmap, 0);


        gl.glEnable(GL10.GL_TEXTURE_2D);                        // 启用纹理映射
        gl.glShadeModel(GL10.GL_SMOOTH);                        // 启用阴影平滑
        gl.glClearColor(0.0f, 0.0f, 0.0f, 0.5f);                // 黑色背景
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);
        // 真正精细的透视修正
        gl.glBlendFunc(GL10.GL_SRC_ALPHA,GL10.GL_ONE);// 设置混色函数取得半透明效果
        gl.glEnable(GL10.GL_BLEND);                        // 启用混色

        for (int i=0; i<num; i++)                          // 创建循环设置全部星星
        {
            Star starTMP = new Star();
            starTMP.angle=0.0f;                            // 所有星星都从零角度开始
            starTMP.dist=((float)(i)/(float)num)*5.0f; // 计算星星离中心的距离
            starTMP.r=random.nextInt(256);      // 为 star[loop]设置随机红色分量
            starTMP.g=random.nextInt(256);      // 为 star[loop]设置随机红色分量
            starTMP.b=random.nextInt(256);      // 为 star[loop]设置随机红色分量
            star[i] = starTMP;
        }
    }

    public boolean onKeyUp(int keyCode, KeyEvent event)
    {
        twinkle=!twinkle;
        return false;
    }
}
class Star
{
    int     r, g, b;    // 星星的颜色
    float   dist;       // 星星距离中心的距离
    float   angle   = 0.0f;// 当前星星所处的角度
}
```

至此整个程序编写完毕，执行后的效果如图 16-1 所示。

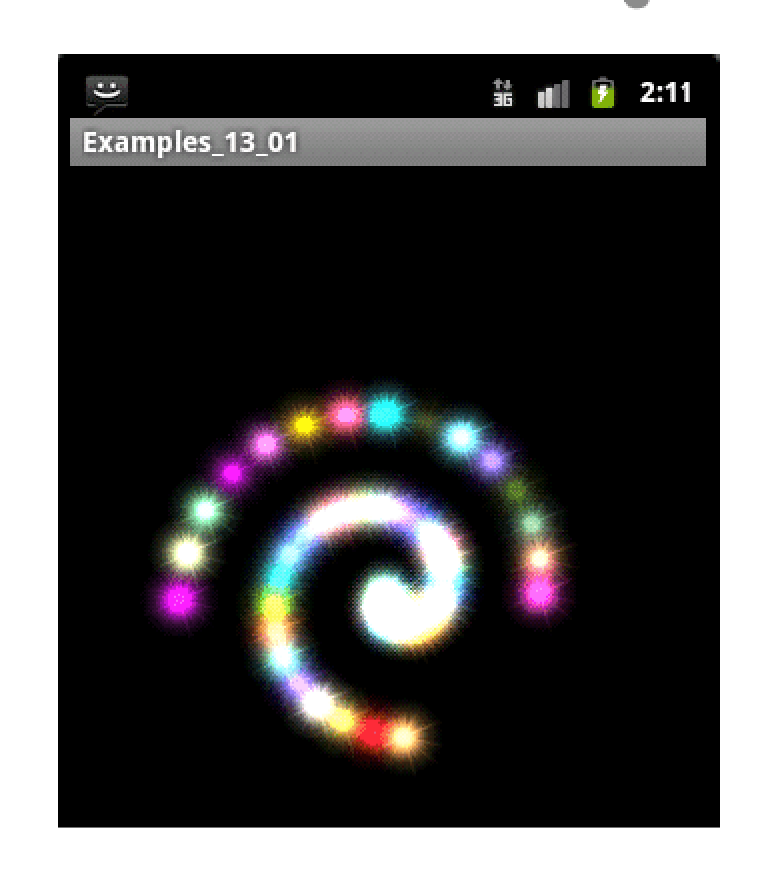


图 16-1　执行效果

16.2.2　模拟一个 3D 场景

题　目	目　的	源码路径
题目 2	构建一个模拟 3D 场景效果	“光盘:\daima\16\example_13_02” 文件夹

我知道任何一个复杂的对象都是由一些简单的形状构成的，所以在创建复杂环境之前应该先定义一个场景的数据结构，例如，可以用 3D 空间中的坐标值(x, y, z)以及纹理坐标(u, v)来定义一个三角形的顶点。下面是具体实现流程。

(1)　在布局文件中设置插入一个 TextView。

(2)　编写主程序文件 ScData.java，用于设置顶点结构 Vertex 和三角形结构 Triangle。具体实现代码如下：

```
//VERTEX 顶点结构
class VERTEX
{
    float x, y, z;  // 3D 坐标
    float u, v;     // 纹理坐标
    public VERTEX(float x,float y,float z,float u,float v)
    {
        this.x = x;
        this.y = y;
        this.z = z;
        this.u = u;
        this.v = v;
    }
}
//TRIANGLE 三角形结构
class TRIANGLE
{
    // VERTEX 矢量数组，大小为 3
    VERTEX[]    vertex  = new VERTEX[3];
}
//SECTOR 区段结构
class SECTOR
{
```

```
    // Sector 中的三角形个数
    int numtriangles;
    // 三角形的 list
    List<TRIANGLE>  triangle    = new ArrayList<TRIANGLE>();
}
```

(3) 绘制 3D 场景。具体代码如下：

```
        for(TRIANGLE triangle : sector1.triangle)
        {
         vertexPointer.clear();
         texCoordPointer.clear();
         gl.glNormal3f(0.0f, 0.0f, 1.0f);
         for(int i=0; i<3; i++)
         {
             VERTEX vt = triangle.vertex[i];
             vertexPointer.put(vt.x);
             vertexPointer.put(vt.y);
             vertexPointer.put(vt.z);
             texCoordPointer.put(vt.u);
             texCoordPointer.put(vt.v);
         }
         gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 4);
        }

       gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
       gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
```

(4) 定义方法 SetupWorld()，用于读取资源数据，去掉每个三角形的顶点数据。具体代码如下：

```
    //读取资源数据
    public void SetupWorld()
    {
        BufferedReader br = new BufferedReader(new InputStreamReader
        (GLFile.getFile("data/world.txt")));

        TRIANGLE triangle = new TRIANGLE();
        int vertexIndex = 0;

        try {
            String line = null;
            while((line = br.readLine()) != null){
                if(line.trim().length() <= 0 || line.startsWith("/")){
                    continue;
                }
                String part[] = line.trim().split("\\s+");
                float x = Float.valueOf(part[0]);
                float y = Float.valueOf(part[1]);
                float z = Float.valueOf(part[2]);
                float u = Float.valueOf(part[3]);
                float v = Float.valueOf(part[4]);
                VERTEX vertex = new VERTEX(x, y, z, u, v);
```

```
                triangle.vertex[vertexIndex] = vertex;

                vertexIndex ++;
                if(vertexIndex == 3){
                    vertexIndex = 0;
                    sector1.triangle.add(triangle);
                    triangle = new TRIANGLE();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

(5)　定义方法 onKeyUp 来响应键盘按键事件。具体代码如下：

```
    public boolean onKeyUp(int keyCode, KeyEvent event)
    {
        switch ( keyCode )
        {
            case KeyEvent.KEYCODE_DPAD_LEFT:
                yrot -= 1.5f;
                break;
            case KeyEvent.KEYCODE_DPAD_RIGHT:
                yrot += 1.5f;
                break;
            case KeyEvent.KEYCODE_DPAD_UP:
                // 沿游戏者所在的 X 平面移动
                xpos -= (float)Math.sin(heading*piover180) * 0.05f;
                // 沿游戏者所在的 Z 平面移动
                zpos -= (float)Math.cos(heading*piover180) * 0.05f;

                if (walkbiasangle >= 359.0f)    // 如果 walkbiasangle 大于 359 度
                {
                    walkbiasangle = 0.0f;       // 将 walkbiasangle 设为 0
                }
                else
                {
                     walkbiasangle+= 10;// 如果 walkbiasangle < 359 ，则增加 10
                }
                                        // 使游戏者产生跳跃感
                walkbias = (float)Math.sin(walkbiasangle * piover180)/20.0f;

                break;
            case KeyEvent.KEYCODE_DPAD_DOWN:
                // 沿游戏者所在的 X 平面移动
                xpos += (float)Math.sin(heading*piover180) * 0.05f;
                // 沿游戏者所在的 Z 平面移动
                zpos += (float)Math.cos(heading*piover180) * 0.05f;
                // 如果 walkbiasangle 小于 1 度
                if (walkbiasangle <= 1.0f)
                {
```

```
                walkbiasangle = 359.0f; // 使 walkbiasangle 等于 359

            }
            else
            {
                walkbiasangle-= 10;     // 如果 walkbiasangle > 1 减去 10
            }
                                        // 使游戏者产生跳跃感
            walkbias = (float)Math.sin(walkbiasangle * piover180)/20.0f;

            break;
        }
        return false;
    }
```

至此整个程序编写完毕，执行后的效果如图 16-2 所示。

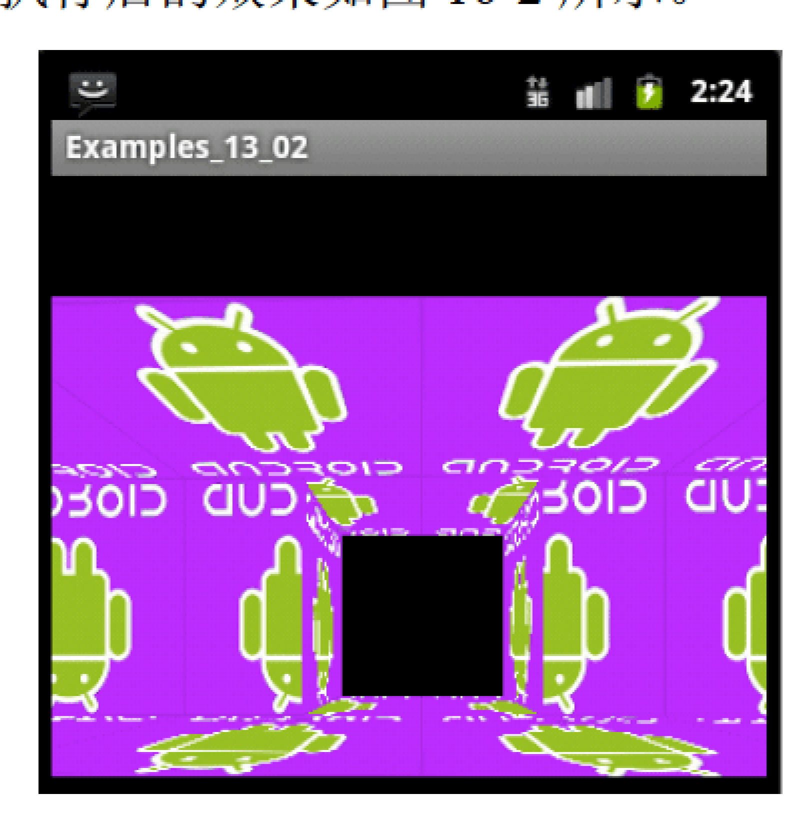


图 16-2　执行效果

16.2.3　浮动的旗帜

题　目	目　的	源码路径
题目 3	实现一个浮动的旗帜效果	“光盘:\daima\16\example3”文件夹

本题目应该结合正弦波动，这样图像看起来像浮动的样子。需要用单独的数组来存放网格各顶点独立的 x，y，z 坐标。下面是具体的实现流程。

(1)　在布局文件中插入一个 TextView。

(2)　使用数组 Vertex 来保存网格各顶点独立的 x，y，z 坐标，在此我设置用 45×45 点形成。具体实现代码如下：

```
float vertex[][][]=new float[45][45][3];    // Points 网格顶点数组(45,45,3)
int wiggle_count = 0;                       // 指定旗形波浪的运动速度
float hold;                                 // 临时变量
float xrot, yrot, zrot;
int texture = -1;
FloatBuffer texCoord = FloatBuffer.allocate(8);
FloatBuffer points = FloatBuffer.allocate(12);
```

(3) 通过两个循环来初始化网格上的点。具体代码如下：

```
// 沿 X 平面循环
for(int x=0; x<45; x++)
{
    // 沿 Y 平面循环
    for(int y=0; y<45; y++)
    {
        // 向表面添加波浪效果
        vertex[x][y][0]=((float)x/5.0f)-4.5f;
        vertex[x][y][1]=(((float)y/5.0f)-4.5f);
        vertex[x][y][2]=(float)(Math.sin(((((float)x/5.0f)*40.0f)/360.0f)*3.
        141592654*2.0f));
    }
}
```

(4) 开始绘制图形，通过一个双循环来处理每个面的顶点和纹理。具体代码如下：

```
for( x = 0; x < 44; x++ ){
    for( y = 0; y < 44; y++) {
         float_x = (float)(x)/44.0f;        // 生成 X 浮点值
         float_y = (float)(y)/44.0f;        // 生成 Y 浮点值
         float_xb = (float)(x+1)/44.0f;     // X 浮点值+0.0227f
         float_yb = (float)(y+1)/44.0f;     // Y 浮点值+0.0227f

        texCoord.clear();
        texCoord.put(float_x);
        texCoord.put(float_y);
        texCoord.put(float_x);
        texCoord.put(float_yb);
        texCoord.put(float_xb);
        texCoord.put(float_yb);
        texCoord.put(float_xb);
        texCoord.put(float_y);

        points.clear();
        points.put(vertex[x][y][0]);
        points.put(vertex[x][y][1]);
        points.put(vertex[x][y][2]);

        points.put(vertex[x][y+1][0]);
        points.put(vertex[x][y+1][1]);
        points.put(vertex[x][y+1][2]);

        points.put(vertex[x+1][y+1][0]);
        points.put(vertex[x+1][y+1][1]);
        points.put(vertex[x+1][y+1][2]);

        points.put(vertex[x+1][y][0]);
        points.put(vertex[x+1][y][1]);
        points.put(vertex[x+1][y][2]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 0, 4);
```

(5) 因为要实现波浪效果，所以要绘制两次场景，生成波浪效果的具体代码如下：

```
if( wiggle_count == 2 )                       // 用来降低波浪速度(每隔 2 帧一次)
{
    for( y = 0; y < 45; y++ )                 // 沿 Y 平面循环
    {
        hold=vertex[0][y][2];                 // 存储当前左侧波浪值
        for( x = 0; x < 44; x++)              // 沿 X 平面循环
        {
            // 当前波浪值等于其右侧的波浪值
            vertex[x][y][2] = vertex[x+1][y][2];
        }
        vertex[44][y][2]=hold;                // 刚才的值成为最左侧的波浪值
    }
    wiggle_count = 0;                         // 计数器清零
```

至此，整个程序编写完毕，执行后的效果如图 16-3 所示。

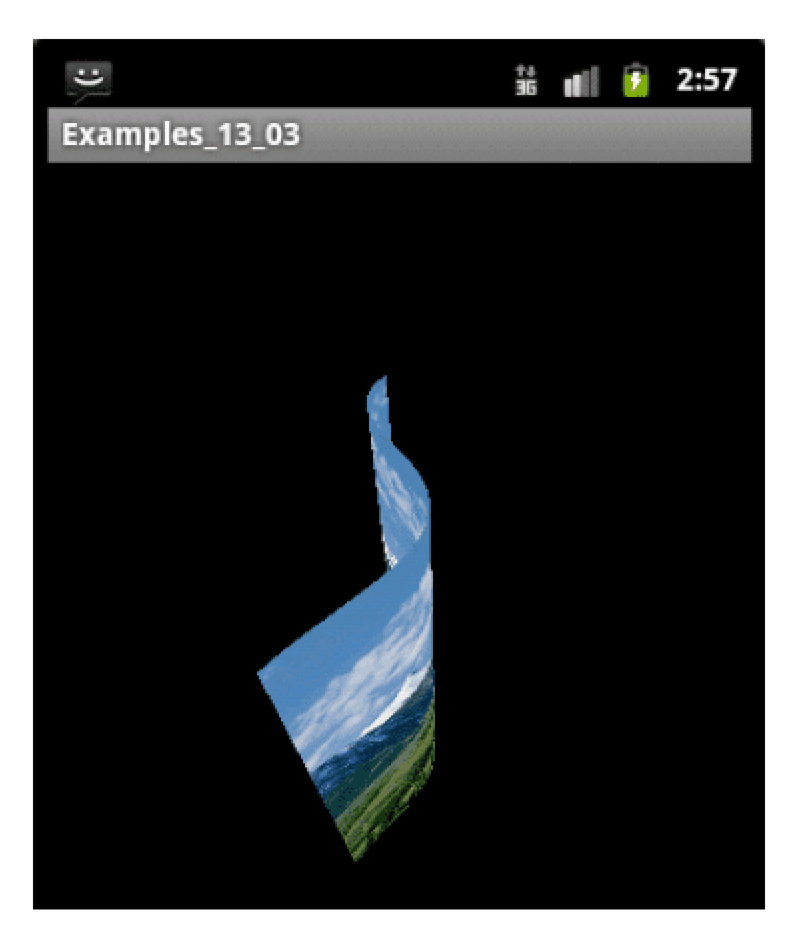


图 16-3 执行效果

16.2.4 列表显示多个物体

题 目	目 的	源码路径
题目 4	通过列表来显示多个类似的 3D 物体	“光盘:\daima\16\example_13_04”文件夹

根据我原来掌握的知识，在场景中显示多个类似物体的方法是通过循环来实现的。但是，这样会增加代码量并使得程序运行缓慢，最好用列表来实现此类功能，通过列表不但能够减少代码数量，而且能够减少代码的长度。下面是具体的实现流程。

(1) 在布局文件中插入一个 TextView。

(2) 定义 FloatBuffer 类型变量来保存列表数据。具体实现代码如下：

```
// 保存盒子的显示列表
FloatBuffer boxVertices = FloatBuffer.allocate(60);
FloatBuffer boxTexCoords= FloatBuffer.allocate(40);
// 保存盒子顶部的显示列表
FloatBuffer topVertices = FloatBuffer.allocate(12);
```

```
FloatBuffer topTexCoords= FloatBuffer.allocate(8);
```

(3) 定义两个数组分别来存储一个立方体盒子的颜色，即顶部颜色和盒子颜色。具体实现代码如下：

```
float[][] boxcol = {
        {1.0f, 0.0f, 0.0f},
        {1.0f, 0.5f, 0.0f},
        {1.0f, 1.0f, 0.0f},
        {0.0f, 1.0f, 0.0f},
        {0.0f, 1.0f, 1.0f},
};

float[][] topcol= {
        {0.5f, 0.0f, 0.0f},
        {0.5f, 0.25f, 0.0f},
        {0.5f, 0.5f, 0.0f},
        {0.0f, 0.5f, 0.0f},
        {0.0f, 0.5f, 0.5f},
};
```

(4) 初始化列表数据，保存了一个立方体的顶点和贴图数据，在绘制时循环绘制这些立方体。具体实现代码如下：

```
public void BuildLists(GL10 gl)
{
    boxTexCoords.put(new float[]{1.0f, 1.0f,0.0f, 1.0f,0.0f, 0.0f,1.0f,0.0f});
   boxVertices.put(new float[]{-1.0f, -1.0f, -1.0f,1.0f, -1.0f, -1.0f,1.0f,
    -1.0f,  1.0f,-1.0f, -1.0f,  1.0f});

    boxTexCoords.put(new float[]{0.0f, 0.0f,1.0f, 0.0f,1.0f, 1.0f,0.0f, 1.0f});
   boxVertices.put(new float[]{-1.0f, -1.0f,  1.0f,1.0f, -1.0f,  1.0f,1.0f,
    1.0f,  1.0f,-1.0f,  1.0f,  1.0f});

    boxTexCoords.put(new float[]{1.0f, 0.0f,1.0f, 1.0f,0.0f, 1.0f,0.0f, 0.0f});
    boxVertices.put(new float[]{-1.0f, -1.0f, -1.0f,-1.0f,  1.0f,
    -1.0f,1.0f,  1.0f, -1.0f,1.0f, -1.0f, -1.0f});

    boxTexCoords.put(new float[]{1.0f, 0.0f,1.0f, 1.0f,0.0f, 1.0f,0.0f, 0.0f});
    boxVertices.put(new float[]{1.0f, -1.0f, -1.0f,1.0f,  1.0f, -1.0f,1.0f,
    1.0f,  1.0f,1.0f, -1.0f,  1.0f});

    boxTexCoords.put(new float[]{0.0f, 0.0f,1.0f, 0.0f,1.0f, 1.0f,0.0f, 1.0f});
    boxVertices.put(new float[]{-1.0f, -1.0f, -1.0f,-1.0f, -1.0f, 1.0f,
    -1.0f, 1.0f, 1.0f,-1.0f, 1.0f, -1.0f});
    topTexCoords.put(new float[]{0.0f, 1.0f,0.0f, 0.0f,1.0f, 0.0f,1.0f, 1.0f});
    topVertices.put(new float[]{-1.0f, 1.0f, -1.0f,-1.0f, 1.0f, 1.0f,1.0f,
    1.0f, 1.0f,1.0f, 1.0f, -1.0f});
}
```

(5) 绘制 15 个立方体盒子并实现堆积显示，即在绘制每一个立方体之前将其设置到相应的位置，此功能是通过函数 onDrawFrame 实现的。具体实现代码如下：

```
public void onDrawFrame(GL10 gl)
{
    // 清除屏幕和深度缓存
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);


    // 绑定纹理
    gl.glBindTexture(GL10.GL_TEXTURE_2D, texture);

    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);

    for (yloop=1;yloop<6;yloop++)
    {
        for (xloop=0;xloop<yloop;xloop++)
        {
            gl.glLoadIdentity();            // 重置模型变化矩阵

            // 设置盒子的位置
            gl.glTranslatef(1.4f+((float)(xloop)*2.8f)-((float)(yloop)
              *1.4f),((6.0f-(float)(yloop))*2.4f)-7.0f,-20.0f);

            gl.glRotatef(45.0f-(2.0f*yloop)+xrot,1.0f,0.0f,0.0f);

            gl.glRotatef(45.0f+yrot,0.0f,1.0f,0.0f);

            gl.glColor4f(boxcol[yloop-1][0], boxcol[yloop-1][1], boxcol
              [yloop-1][2], 1.0f);

            gl.glVertexPointer(3, GL10.GL_FLOAT, 0, boxVertices);
            gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, boxTexCoords);
            gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 0, 4);
            gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 4, 4);
            gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 8, 4);
            gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 12, 4);
            gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 16, 4);

          /* Select The Top Color */
            gl.glColor4f(topcol[yloop-1][0], topcol[yloop-1][1],
              topcol[yloop-1][2], 1.0f);
            gl.glVertexPointer(3, GL10.GL_FLOAT, 0, topVertices);
            gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, topTexCoords);
            gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 0, 4);
        }
    }

   gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
   gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);

   xrot+=0.5f;
   yrot+=0.6f;
```

```
        zrot+=0.3f;
    }
```

至此，整个程序编写完毕，执行后的效果如图 16-4 所示。

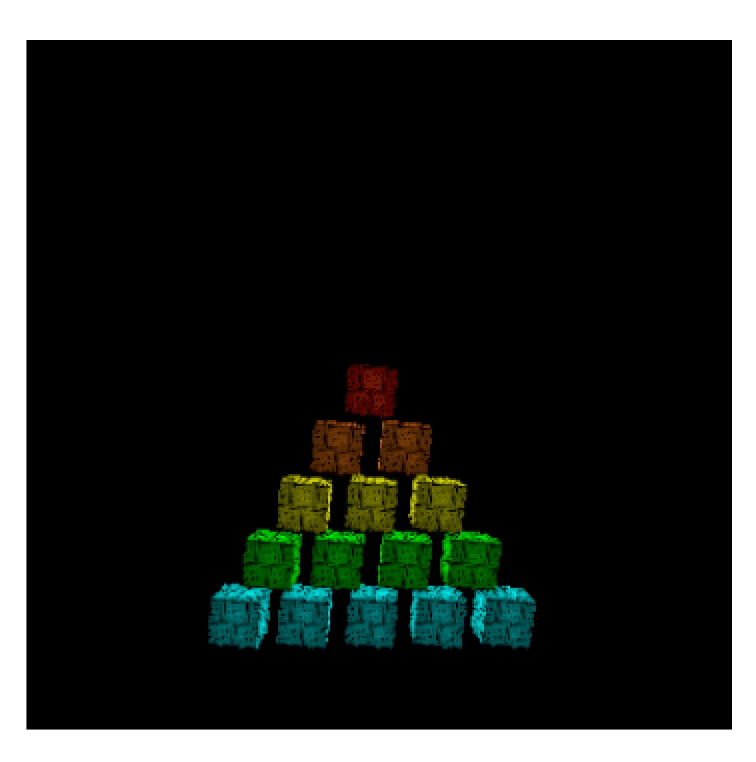


图 16-4　执行效果

16.2.5　粒子系统

题　目	目　的	源码路径
题目 5	模拟实现粒子特效	“光盘:\daima\16\example_13_06”文件夹

粒子系统对我来说很熟悉了，在游戏中体验过很多次粒子系统实现的特效，并且很多特效是利用某一项技能才实现的。本次我准备用三角形来绘制一个粒子效果，先把某一个“点”抽象成一个物体，然后赋予这个物体一些属性。下面是具体的实现流程。

(1)　在布局文件中插入一个 TextView。

(2)　定义类 particle 来表示“点”。具体实现代码如下：

```
public class particle
{
      boolean active; /* Active (Yes/No) */
      float life;   /* Particle Life    */
      float fade;   /* Fade Speed       */

      float r;      /* Red Value        */
      float g;      /* Green Value      */
      float b;      /* Blue Value       */

      float x;      /* X Position       */
      float y;      /* Y Position       */
      float z;      /* Z Position       */

      float xi;     /* X Direction      */
      float yi;     /* Y Direction      */
      float zi;     /* Z Direction      */

      float xg;     /* X Gravity        */
      float yg;     /* Y Gravity        */
      float zg;     /* Z Gravity        */
}
```

(3) 为了更好地操作和控制微粒，加入了如下变量，代码如下：

```
public final static int MAX_PARTICLES =1000;
boolean rainbow=true;  /* Toggle rainbow effect */
Random random = new Random();
float slowdown=0.5f;    /* 减速例子*/
float xspeed=1;         /* x 方向的速度*/
float yspeed=3;         /* y 方向的速度*/
float zoom=-30.0f;      /* 沿 z 轴缩放*/

int loop;               /* 循环变量*/
int col=0;              /* 当前的颜色*/
int delay;              /* 延迟彩虹效果*/
```

(4) 定义数组 colors，用于存储 12 种不同的颜色。具体代码如下：

```
static float colors[][]=
{
  {1.0f,  0.5f,  0.5f},
  {1.0f,  0.75f, 0.5f},
  {1.0f,  1.0f,  0.5f},
  {0.75f, 1.0f,  0.5f},
  {0.5f,  1.0f,  0.5f},
  {0.5f,  1.0f,  0.75f},
  {0.5f,  1.0f,  1.0f},
  {0.5f,  0.75f, 1.0f},
  {0.5f,  0.5f,  1.0f},
  {0.75f, 0.5f,  1.0f},
  {1.0f,  0.5f,  1.0f},
  {1.0f,  0.5f,  0.75f}
};
```

(5) 装载纹理贴图，实现初始化处理。具体代码如下：

```
public void ResetParticle(int num, int color, float xDir, float yDir, float
  zDir)
{
    particle tmp = new particle();
    /* Make the particels active */
    tmp.active=true;
    /* Give the particles life */
    tmp.life=1.0f;
    /* Random Fade Speed */
    tmp.fade=(float)(rand()%100)/1000.0f+0.003f;
    /* Select Red Rainbow Color */
    tmp.r=colors[color][0];
    /* Select Green Rainbow Color */
    tmp.g=colors[color][1];
    /* Select Blue Rainbow Color */
    tmp.b=colors[color][2];
    /* Set the position on the X axis */
    tmp.x=0.0f;
    /* Set the position on the Y axis */
```

```
        tmp.y=0.0f;
        /* Set the position on the Z axis */
        tmp.z=0.0f;
        /* Random Speed On X Axis */
        tmp.xi=xDir;
        /* Random Speed On Y Axi */
        tmp.yi=yDir;
        /* Random Speed On Z Axis */
        tmp.zi=zDir;
        /* Set Horizontal Pull To Zero */
        tmp.xg=0.0f;
        /* Set Vertical Pull Downward */
        tmp.yg=-0.5f;
        /* Set Pull On Z Axis To Zero */
        tmp.zg=0.0f;
        particles[num] = tmp;
        return;
    }
```

(6) 为粒子分配一种颜色，通过方法 onDrawFrame 实现对粒子的绘制处理。具体代码如下：

```
    public void onDrawFrame(GL10 gl)
    {
        FloatBuffer vertices = FloatBuffer.wrap(new float[12]);
        FloatBuffer texcoords = FloatBuffer.wrap(new float[8]);

        /* Clear The Screen And The Depth Buffer */
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        /* Enable vertices and texcoords arrays */
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);

        /* Set pointers to vertices and texcoords */
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertices);
        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, texcoords);

        gl.glLoadIdentity();

        /* Modify each of the particles */
        for (loop = 0; loop < MAX_PARTICLES; loop++)
        {
            if (particles[loop].active)
            {
                /* Grab Our Particle X Position */
                float x = particles[loop].x;

                /* Grab Our Particle Y Position */
                float y = particles[loop].y;

                /* Particle Z Position + Zoom */
                float z = particles[loop].z + zoom;
```

```
        /*
         * Draw The Particle Using Our RGB Values, Fade The Particle
         * Based On It's Life
         */
        gl.glColor4f(particles[loop].r, particles[loop].g, particles[loop].b,
        particles[loop].life);

        texcoords.clear();
        vertices.clear();
        /* Top Right */
        texcoords.put(1.0f);
        texcoords.put(1.0f);
        vertices.put(x + 0.5f);
        vertices.put(y + 0.5f);
        vertices.put(z);

        /* Top Left */
        texcoords.put(0.0f);
        texcoords.put(1.0f);
        vertices.put(x - 0.5f);
        vertices.put(y + 0.5f);
        vertices.put(z);

        /* Bottom Right */
        texcoords.put(1.0f);
        texcoords.put(0.0f);
        vertices.put(x + 0.5f);
        vertices.put(y - 0.5f);
        vertices.put(z);

        /* Bottom Left */
        texcoords.put(0.0f);
        texcoords.put(0.0f);
        vertices.put(x - 0.5f);
        vertices.put(y - 0.5f);
        vertices.put(z);

        /* Build Quad From A Triangle Strip */
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);

        /* Move On The X Axis By X Speed */
        particles[loop].x += particles[loop].xi / (slowdown * 1000);
        /* Move On The Y Axis By Y Speed */
        particles[loop].y += particles[loop].yi / (slowdown * 1000);
        /* Move On The Z Axis By Z Speed */
        particles[loop].z += particles[loop].zi / (slowdown * 1000);

        /* Take Pull On X Axis Into Account */
        particles[loop].xi += particles[loop].xg;
```

```
                /* Take Pull On Y Axis Into Account */
                particles[loop].yi += particles[loop].yg;
                /* Take Pull On Z Axis Into Account */
                particles[loop].zi += particles[loop].zg;

                /* Reduce Particles Life By 'Fade' */
                particles[loop].life -= particles[loop].fade;

                /* If the particle dies, revive it */
                if (particles[loop].life < 0.0f)
                {
                    float xi, yi, zi;
                    xi = xspeed + (float) ((rand() % 60) - 32.0f);
                    yi = yspeed + (float) ((rand() % 60) - 30.0f);
                    zi = (float) ((rand() % 60) - 30.0f);
                    ResetParticle(loop, col, xi, yi, zi);
                }
            }
        }
          /* Disable texcoords and vertices arrays */
        gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);

        /* Draw it to the screen */
        gl.glFinish();
    }
```

至此，整个程序编写完毕，执行后的效果如图 16-5 所示。

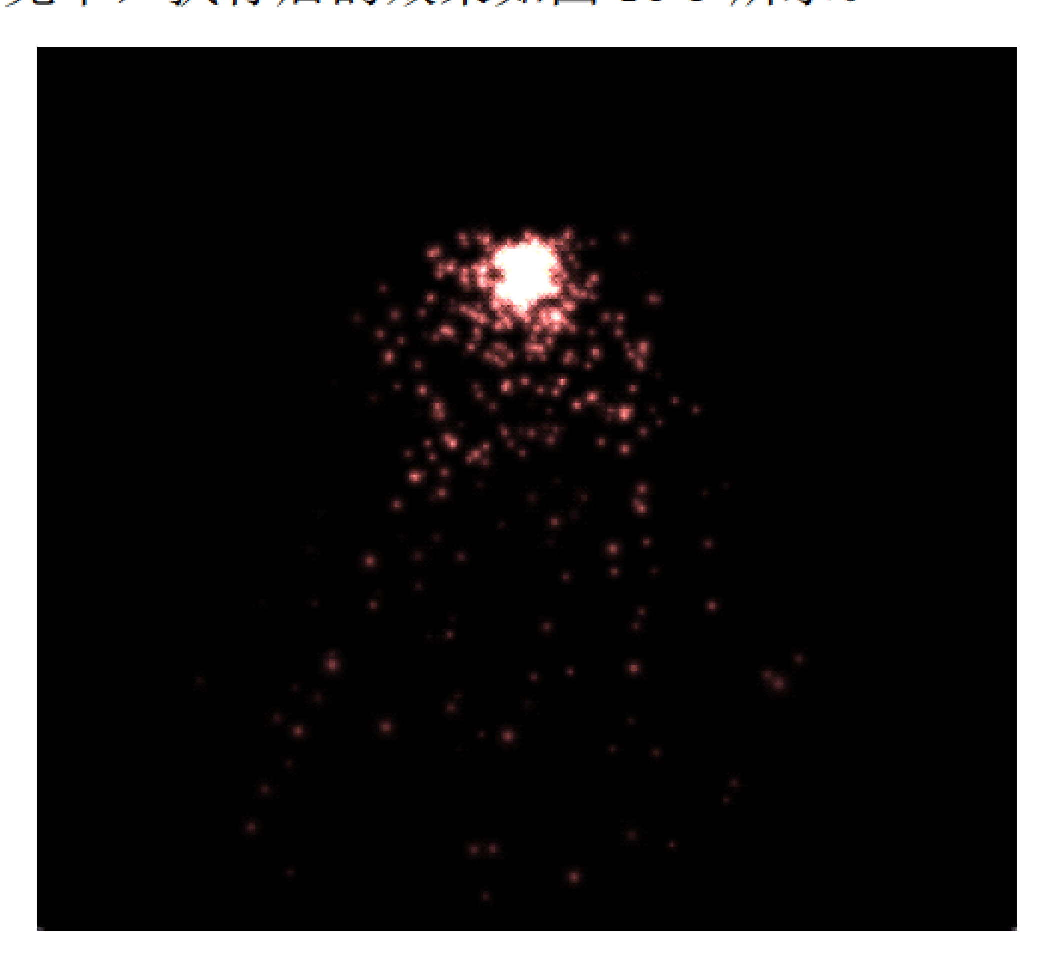


图 16-5　执行效果



第五篇 综合实战篇

第 17 章 使用 Google API

Android 作为谷歌旗下的产品，它能够使用官方很多强大的服务功能，例如，Google API、日历、相册和文件等。在本章内容中，将通过几个典型实例的实现过程，来详细介绍 Android 和其官方服务相结合的基本知识和具体使用流程。

17.1 模拟验证官方账号

题 目	目 的	源码路径
演练 1	通过账号验证来获取 Google Account Authentication Service 所发出的凭据	“光盘:\daima\17\example1”文件夹

17.1.1 Google Accout

在现实应用中，基于 Google 的大多数服务都需要通过官方进行账号验证。当前的 Google Account 已经阵容强大，由单一的 AuthSub 发展到 Oauth、Federated、Hybird 四种，具体说明如下。

(1) AuthSub：提供单独的 Google Service Access，比如 Gmail Contact。如果你要一次使用多个 Google Service，比如，需要同时读取 Picasaweb 和 Youtube 的数据，那么用 Authsub 就不那么合适了，因为你需要让用户登录 Google 两次。

(2) OAuth：和 Authsub 相比，提供了 sign-on，一次登录取得的 Authtoken 可以成功获得几个 Google Services。

(3) Federated：本质上是 Openid,，它是唯一可以让你在使用 Google Account 登录自己网站的同时，拿到用户电子邮件的方法。

(4) Hybird：集成了 Oauth 和 Openid，让你的网站可以同时拿到 Google Service 访问权限，以及用户的电子邮件。

由此可见，AuthSub 即将过时，现在的好处是使用 Authsub 不需要在 Google 注册自己的网站。

17.1.2 具体实现

编写的主文件是 example1.java、example1_01_02.java 和 GoogleAuthSub.java。

1. 文件 example1.java

文件 example1.java 功能是进行登录 UI 和获取用户账号的密码。首先，通过 TextView 的 onClick()事件为起点，调用自定义的 showLoginForm()方法显示登录表单。具体实现代码如下：

```
public class example1 extends Activity
{
  /*声明变量*/
  private TextView mTextView01;
  private LayoutInflater mInflater01;
  private View mView01;
  private EditText mEditText01,mEditText02;
  private String TAG = "HIPPO_DEBUG";
  /* 中文字的间距 */
  private int intShiftPadding = 14;
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /* 创建 DisplayMetrics 对象，取得屏幕分辨率 */
    DisplayMetrics dm = new DisplayMetrics();
    getWindowManager().getDefaultDisplay().getMetrics(dm);
    mTextView01 = (TextView)findViewById(R.id.myTextView1);
    /* 将文字 Label 放在屏幕右上方 */
    mTextView01.setLayoutParams
    (
      new AbsoluteLayout.LayoutParams(intShiftPadding*mTextView01.getText().
      toString().length(),18,(dm.widthPixels-(intShiftPadding*mTextView01.
      getText().toString().length()))-10,0)
    );
    /* 处理用户点击 TextView 文字的事件处理 -登录*/
    mTextView01.setOnClickListener(new TextView.OnClickListener()
    {
      @Override
      public void onClick(View v)
      {
        // TODO Auto-generated method stub

        /* 显示登录对话框 */
        showLoginForm();
```

```
    }
  });
}
/* 自定义登录对话框函数 */
private void showLoginForm()
{
  try
  { /* 以 LayoutInflater 取得主 Activity 的 context */
    mInflater01 = LayoutInflater.from(example1.this);
    /* 设置创建的 View 所要使用的 Layout Resource */
    mView01 = mInflater01.inflate(R.layout.login, null);
    /* 账号 EditText */
    mEditText01=(EditText)mView01.findViewById(R.id.myEditText1);
    /* 密码 EditText */
    mEditText02=(EditText)mView01.findViewById(R.id.myEditText2);
    /*创建 AlertDialog 窗口来取得用户账号密码*/
    new AlertDialog.Builder(this)
    .setView(mView01)
    .setPositiveButton("OK",
    new DialogInterface.OnClickListener()
    {
      /*覆盖 onClick()来触发取得 Token 事件与完成登录事件*/
      public void onClick(DialogInterface dialog, int whichButton)
      {
        if(processGoogleLogin(mEditText01.getText().toString(),
        mEditText02.getText().toString()))
        {
          Intent i = new Intent();
          /*登录后调用注销程序(EX09_01_02.java)*/
          i.setClass(example1.this, example1_01_02.class);
          startActivity(i);
          finish();
        }
      }
    }).show();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}
/*调用 GoogleAuthSub 来取得 Google 账号的 Authentication Token*/
private boolean processGoogleLogin(String strUID, String strUPW)
{
  try
  {
    /*建构自定义的 GoogtleAuthSub 对象*/
    GoogleAuthSub gas = new GoogleAuthSub(strUID, strUPW);
    /*取得 Google Token*/
    String strAuth =  gas.getAuthSubToken();
    /*将取回的 Google Token 写入 log 中*/
    Log.i(TAG, strAuth);
```

```
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    return true;
  }
}
```

showLoginForm()方法是一个使用 LayoutInflater 获取主 Activity 的 context，搭配 AlertDialog 所构建的 Login Form 表单。当用户输入账号和密码后，开始重写 DialogInterface.OnClickListener() 的 onClick()事件来调用自定义的 processGoogleLogin 处理和 Google 账号验证的连接事件。当通过 Google 验证后取得 Google Authentication Token，通过 Intent 打开 example1_01_02.java 以改变 UI 的状态。

2. 文件 example1_01_02.java

文件 example1_01_02.java 的功能是注销返回登录界面的工作，即将原来的登录状态改为注销状态，并实现 TextView 的 onClick()方法。当用户单击 TextView 后，则通过自定义的 Intent 来调用 example.java，返回到程序的等待状态。具体实现代码如下：

```
public class example1_01_02 extends Activity
{
  private TextView mTextView03;
  /* 中文字的间距 */
  private int intShiftPadding = 14;

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    setContentView(R.layout.loginok);

    /* 创建 DisplayMetrics 对象，取得屏幕分辨率 */
    DisplayMetrics dm = new DisplayMetrics();
    getWindowManager().getDefaultDisplay().getMetrics(dm);

    /*通过 findViewById()来取得 TextView 对象*/
    mTextView03 = (TextView)findViewById(R.id.myTextView3);

    /* 将文字 Label 放在屏幕右上方 */
    mTextView03.setLayoutParams
    (
      new AbsoluteLayout.LayoutParams(intShiftPadding*mTextView03.getText().
        toString().length(),18,(dm.widthPixels-(intShiftPadding*mTextView03.
        getText().toString().length()))-10,0)
    );

    /* 处理用户点击 TextView 文字的事件处理-注销 */
```

```
    mTextView03.setOnClickListener(new TextView.OnClickListener()
    {
      /*覆盖 onClick()事件*/
      @Override
      public void onClick(View v)
      {
        // TODO Auto-generated method stub
        Intent i = new Intent();
        /*注销后调用登录程序(EX09_01.java)*/
        i.setClass(example1_01_02.this, example1.class);
        startActivity(i);
        finish();
      }
    });
  }
}
```

3. 文件 GoogleAuthSub.java

文件 GoogleAuthSub.java 的功能是进行 HttpGet 并取得 Google Calendar 的服务 Token 作为示范，此文件是整个实例的核心，通过 Google 提供的 ClientLogin 机制，使用 HttpPost 连接到 https://www.google.com/accounts/ClientLogin，并同时将用户账号和密码及其相关参数以 Name Value Pair 字符串带入，通过自定义的 getAuth()方法获取 Google 认证的 Authentication Token，然后模拟 Google 网络服务 AuthSub 的方法，将自定义的 Header 和 HttpGet 方法带入 Token 来获取用户 Google Calendar 服务中的所有日历数据，并以 xml 文件存储于临时文件中，作为使用 Google 服务的规范。具体实现代码如下：

```
public class GoogleAuthSub
{
  /*声明变量*/
  private DefaultHttpClient httpclient;
  private HttpPost httpost;
  private HttpResponse response;
  private String strGoogleAccount;
  private String strGooglePassword;
  private String TAG = "IRDC_DEBUG";

  /*GoogleAuthSub 对象的构造器*/
  public GoogleAuthSub(String strUID, String strPWD)
  {
    this.strGoogleAccount = strUID;
    this.strGooglePassword = strPWD;
    httpclient = new DefaultHttpClient();
    httpost = new HttpPost("https://www.google.com/accounts/ClientLogin");
  }

  /*取得 Google Token 方法*/
  public String getAuthSubToken()
  {
    /*创建 Name Value Pair 字符串*/
    List <NameValuePair> nvps = new ArrayList <NameValuePair>();
```

```
nvps.add(new BasicNameValuePair("Email", this.strGoogleAccount));
nvps.add(new BasicNameValuePair("Passwd", this.strGooglePassword));
nvps.add(new BasicNameValuePair("source", "MyApiV1"));
nvps.add(new BasicNameValuePair("service", "cl"));
String GoogleLoginAuth="";
try
{
  /*创建 Http Post 连接*/
  httpost.setEntity(new UrlEncodedFormEntity(nvps, HTTP.DEFAULT_CONTENT_
  CHARSET));
  response = httpclient.execute(httpost);
  if( response.getStatusLine().getStatusCode()!=200 )
  {
    return "";
  }
  /*取回 Google Token*/
  InputStream is = response.getEntity().getContent();
  GoogleLoginAuth = getAuth(is);
  /*模拟 HTTP Header*/
  Header[] headers = new BasicHeader[6];
  headers[0] = new BasicHeader("Content-type", "application/x-www-form-
  urlencoded");
  headers[1] = new BasicHeader("Authorization", "GoogleLogin auth=
  \""+GoogleLoginAuth+"\"");
  headers[2] = new BasicHeader("User-Agent", "Java/1.5.0 06");
  headers[3] = new BasicHeader("Accept", "text/html, image/gif, image/jpeg,
  *; q=.2, */*; q=.2");
  headers[4] = new BasicHeader("Connection", "keep-alive");
  /*发出 Http Get 请求登录 Google Calendar 服务*/
  HttpGet httpget;
  String feedUrl2 = "http://www.google.com/calendar/feeds/default/allcalendars/
  full";
  httpget = new HttpGet(feedUrl2);
  httpget.addHeader(headers[0]);
  httpget.addHeader(headers[1]);
  httpget.addHeader(headers[2]);
  httpget.addHeader(headers[3]);
  httpget.addHeader(headers[4]);
  /*取得 Google Calendar 服务应答*/
  response = httpclient.execute(httpget);
  String strTemp01 = convertStreamToString(response.getEntity().getContent());
  Log.i(TAG, strTemp01);
  /*指定暂存盘位置*/
  String strEarthLog = "/sdcard/googleauth.log";
  BufferedWriter bw;
  bw = new BufferedWriter (new FileWriter(strEarthLog));
  /*将取回文件写入暂存盘中*/
  bw.write( strTemp01, 0, strTemp01.length() );
  bw.flush();

}
```

```
    catch (UnsupportedEncodingException e)
    {
      e.printStackTrace();
    }
    catch (ClientProtocolException e)
    {
      e.printStackTrace();
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
    return GoogleLoginAuth;
  }
  /*自定义读取 token 内容的方法*/
  public String getAuth(InputStream is)
  {
    BufferedReader reader = new BufferedReader(new InputStreamReader(is));
    String line = null;
    String strAuth="";
    try
    {
      while ((line = reader.readLine()) != null)
      {
        Log.d(TAG, ": "+line);
        if( line.startsWith("Auth="))
        {
          strAuth=line.substring(5);
          Log.i("auth",": "+strAuth);
        }
      }
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }
    finally
    {
      try
      {
        is.close();
      }
      catch (IOException e)
      {
        e.printStackTrace();
      }
    }
    return strAuth;
```

```
    }
    /*将数据转为字符串方法*/
    public String convertStreamToString(InputStream is)
    {
      BufferedReader reader = new BufferedReader(new InputStreamReader(is));
      StringBuilder sb = new StringBuilder();
      String line = null;
      try
      {
        while ((line = reader.readLine()) != null)
        {
          sb.append(line);
        }
      }
      catch (IOException e)
      {
        e.printStackTrace();
      }
      finally
      {
        try
        {
          is.close();
        }
        catch (IOException e)
        {
          e.printStackTrace();
        }
      }
      return sb.toString();
    }
}
```

至此，整个演练结束，执行后的效果如图 17-1 所示；单击“登录”链接后显示登录表单界面，如图 17-2 所示；输入账号和密码并单击 OK 按钮后，弹出“成功获取 Token”的提示，如图 17-3 所示。

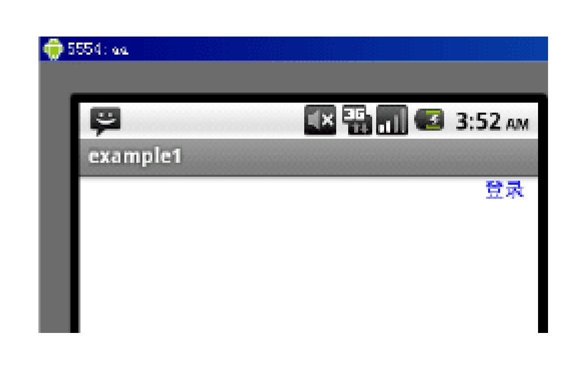


图 17-1　执行效果



图 17-2　登录表单

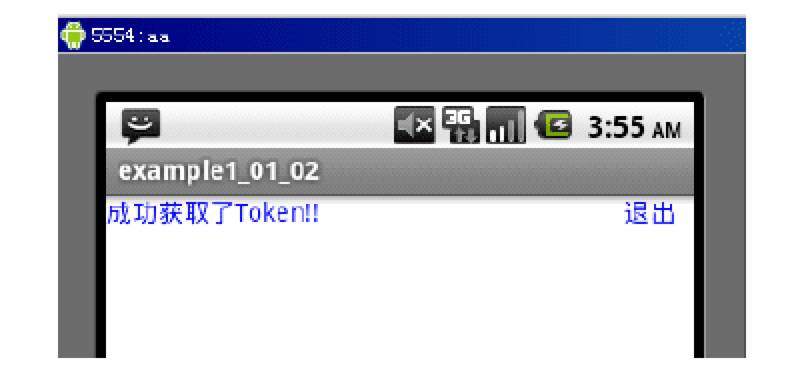


图 17-3　成功获取 Token 的提示

17.2　模拟实现 Google 搜索

题　目	目　的	源码路径
演练 2	通过 Google Search API 实现检索处理	“光盘:\daima\17\example2”文件夹

17.2.1　Google Search API

在现实信息时代，信息已经成为第一生产力。在巨大的网络资源中，检索站点蓬勃发展，百度、雅虎和 Google 都已经站在了时代的最前沿。在 Android 官方服务中，提供了 Google Search API 实现检索处理。

使用 Google Search API 的基本流程如下。

(1)　构造一个搜索服务“容器”。

google.search.SearchControl 实例代表页面上的一个搜索控件，此控件是多种“搜索服务”的“容器”。

(2)　构造一个“搜索服务”对象。

构造函数 google.search.LocalSearch()可以构造一个“搜索服务”对象。“搜索服务”是我自己起的名字，之所以叫这个名字是因为我认为它的核心是服务。尽管默认有一个搜索框和结果列表，但这些都是可以被改变的，甚至搜索结果的内容都是可以改变的，因为我认为这个构造函数的核心是一个“服务”。

(3)　向容器中添加“搜索服务”。

searchControl.addSearcher(searcher, options)。options 的类型为 google.search.SearchOptions。可以添加的搜索服务有多种，到目前为止 Google 提供了以下类型的搜索器：

- LocalSearch；
- WebSearch；
- VideoSearch；
- BlogSearch；
- NewsSearch；
- ImageSearch；
- BookSearch；
- PatentSearch。

(4)　SearchControl 对象调用 draw()方法，按照里面的 DrawOptions 参数画出搜索框，代码如下：

```
searchControl.draw(document.getElementById("from"), drawOptions)
```

上述流程是一般的搜索流程，但 Google 提供了很多选项来定制这些服务，主要选项有两种，一种是 SearcherOptions；另一种是 DrawOptions。

17.2.2　具体实现

本实例的主文件是 example2.java 和 MyAdapter.java，下面分别介绍其实现的流程。

1. 文件 example2.java

文件 example2.java 功能是创建 MyAdapter 对象，此对象是自己实现的 BaseMyAdapter 类。主要实现代码如下：

```
public class example2 extends Activity
{
```

```
  private AutoCompleteTextView myAutoCompleteTextView1;
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    myAutoCompleteTextView1 = (AutoCompleteTextView) findViewById(R.id.
myAutoCompleteTextView1);
    /* new 一个自己实现的 BaseAdapter */
    MyAdapter adapter = new MyAdapter(this);
    myAutoCompleteTextView1.setAdapter(adapter);
  }
}
```

2. 文件 MyAdapter.java

MyAdapter 继承于 BaseAdapter，可以通过覆盖 Filterable 对象中的 getFilter()方法来对输入的关键字进行动态处理。当用户输入关键字时，performFiltering()所返回的 FilterResults 就是查询后的结果。具体实现代码如下：

```
public class MyAdapter extends BaseAdapter implements Filterable
{
  ArrayList<String> keyWordValue = new ArrayList<String>();
  ArrayList<String> resultValue = new ArrayList<String>();
  private Context mContext;
  LinearLayout.LayoutParams param1;
  public MyAdapter(Context context)
  {
    mContext = context;
    param1 = new LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.WRAP_CONTENT,
        LinearLayout.LayoutParams.WRAP_CONTENT);
  }

  @Override
  public int getCount()
  {
    return keyWordValue.size();
  }

  @Override
  public Object getItem(int position)
  {
    return keyWordValue.get(position);
  }

  @Override
  public long getItemId(int position)
  {
    return position;
  }
```

```
@Override
public View getView(int position, View view, ViewGroup viewGroup)
{
  LinearLayout myLinearLayout = new LinearLayout(mContext);
  myLinearLayout.setOrientation(LinearLayout.HORIZONTAL);
  if (position >= keyWordValue.size())
    return myLinearLayout;
  /* 第一个 TextView 放关键字 */
  TextView keyWordTextView = new TextView(this.mContext);
  keyWordTextView.setTextColor(mContext.getResources().getColor( R.drawable.blue));
  keyWordTextView.setTextSize(18);
  keyWordTextView.setWidth(180);
  try
  {
    keyWordTextView
        .setText(keyWordValue.get(position).toString());
  } catch (java.lang.IndexOutOfBoundsException i)
  {
    keyWordTextView.setText("");
  }
  /* 第二个 TextView 放关键字结果数量 */
  TextView resultTextView = new TextView(this.mContext);
  resultTextView.setTextColor(mContext.getResources().getColor(
      R.drawable.red));
  resultTextView.setTextSize(18);
  try
  {
    resultTextView.setText(resultValue.get(position).toString());
  } catch (java.lang.IndexOutOfBoundsException i)
  {
    resultTextView.setText("");
  }
  myLinearLayout.addView(keyWordTextView, param1);
  myLinearLayout.addView(resultTextView, param1);

  return myLinearLayout;
}
@Override
public Filter getFilter()
{
  // TODO Auto-generated method stub
  Filter myFilter = new Filter()
  {

    @Override
    protected FilterResults performFiltering(CharSequence text)
    {

      FilterResults fr = new FilterResults();
      keyWordValue = new java.util.ArrayList<String>();
      resultValue = new java.util.ArrayList<String>();
```

```
      if (text == null || text.length() == 0)
      {
        fr.count = keyWordValue.size();
        fr.values = keyWordValue;
        return fr;
      }

      /* 输入关键字后调用 google 关键字 API */
      changeResult(getGoogleAPI(text.toString()));

      fr.count = keyWordValue.size();
      fr.values = keyWordValue;
      return fr;
    }

    @Override
    protected void publishResults(CharSequence text,
        FilterResults filterResults)
    {
      if (filterResults != null && filterResults.count > 0)
        notifyDataSetChanged();
      else
        notifyDataSetInvalidated();
    }
  };
  return myFilter;
}

/* 访问 GoogleAPI 取得返回的结果字符串 */
private String getGoogleAPI(String text)
{
  String uri = "";
  try
  {
    /* 输入的字要 encode */
    uri = "http://www.google.com/complete/"
        + "search?hl=en&js=true&qu="
        + URLEncoder.encode(text, "utf-8");
  } catch (UnsupportedEncodingException e1)
  {
    // TODO Auto-generated catch block
    e1.printStackTrace();
  }

  URL googleUrl = null;
  HttpURLConnection conn = null;
  InputStream is = null;
  BufferedReader in = null;
  String resultStr = "";
  /* 取得链接 */
  try
  {
```

```
    googleUrl = new URL(uri);
    /* 打开链接 */
    conn = (HttpURLConnection) googleUrl.openConnection();
    int code = conn.getResponseCode();
    /* 连接 OK 时 */
    if (code == HttpURLConnection.HTTP_OK)
    {
      /* 取得返回的 InputStream */
      is = conn.getInputStream();

      in = new BufferedReader(new InputStreamReader(is));
      String inputLine;
      /* 一行一行读取 */
      while ((inputLine = in.readLine()) != null)
      {
        resultStr += inputLine;
      }

    }
  } catch (IOException e)
  {
    // TODO Auto-generated catch block
    e.printStackTrace();
  } finally
  {
    try
    {
      if (is != null)
        is.close();
      if (conn != null)
        conn.disconnect();
    } catch (Exception e)
    {
    }
  }

  return resultStr;
}

/* 处理返回的字符串变成 ArrayList */
private void changeResult(String text)
{
  String resultStr = "";
  String startSub = "new Array(2, ";
  String endSub = "), new Array";
  int start = text.indexOf(startSub);
  int end = text.indexOf(endSub);
  if (start != -1 && end != -1)
  {
    resultStr = text.substring(start + startSub.length(), end);
    /* 去掉前后的" */
    resultStr = resultStr.substring(1, resultStr.length() - 1);
```

```
    /* 以 ", "来分隔字符串变成字符串数组 */
    String total[] = resultStr.split("\\\", \\\"");
    for (int i = 0; i < total.length / 2; i++)
    {
      keyWordValue.add(total[i * 2]);
      /* 将 results 字符串去掉 */
      resultValue
          .add(total[i * 2 + 1].replaceAll(" results", ""));
    }
  }

 }

}
```

上述代码中，在 MyAdapter 类中重写了 getView()，在其中放了两个 TextView，一个显示关键字，另一个显示结果数量。因为 AutoCompleteTextView 绑定了 MyAdapter，所以当 Adapter 动态向 Google 获取查询结果时，也顺便更新了 AutoCompleteTextView 下拉菜单中的结果。

至此，整个演练结束，执行后的效果如图 17-4 所示。



图 17-4　执行效果

17.3　Geocoder 实现地址查询

题　目	目　的	源码路径
演练 3	通过 Geocoder 实现地址反查询	“光盘:\daima\17\example5”文件夹

17.3.1　Geocoder 服务

Google 提供了一个 Geocoder 服务，在非商用情况下，Geocoder 能够反查 Address 对象服务。那么究竟什么是 Geocoder 呢？简单来说，就是根据某个 Key 寻找地理位置的坐标，这个 Key 可以是地址。近年来 Google Map API 做了很多改进，已经自动整合了中文地址的定位功能，不再需要以前的 mashup 了。以前只有 6 个国家提供了街道级别的定位，Google 从不让人失望。当然每日 50 000 次查询的 limit 还在，但是一般也足够了，何况还有客户端的 built-in cache 改善用户体验。

这个 Key 还可以是 IP 地址，例如做一个 library 可以读取本地数据库转换 IP 为坐标，这个坐标只是城市中心的坐标，不过也一般够用了。同时 library 也将提供基于 http request 的公共服务，地址定位和 IP 定位现在都很容易实现了。

17.3.2 具体实现

本实例的主文件是 example5.java，功能是通过地址来获取 GeoPoint 方法，自定义函数 getGeoByAddress()唯一地传入值字符串，通过传入的地址，以 Geocoder.getFromLocationName()方法来获取从 Google 服务器找到的结果。主要实现代码如下：

```
protected void onCreate(Bundle icicle)
{
  // TODO Auto-generated method stub
  super.onCreate(icicle);
  setContentView(R.layout.main);

  mEditText01 = (EditText)findViewById(R.id.myEditText1);
  mEditText01.setText
  (
    getResources().getText(R.string.str_default_address).toString()
  );

  /* 创建 MapView 对象 */
  mMapView01 = (MapView)findViewById(R.id.myMapView1);
  mMapController01 = mMapView01.getController();
  // 设置 MapView 的显示选项(卫星、街道)
  mMapView01.setSatellite(true);
  mMapView01.setStreetView(true);

  mButton01 = (Button)findViewById(R.id.myButton1);
  mButton01.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View v)
    {
      // TODO Auto-generated method stub
      if(mEditText01.getText().toString()!="")
      {
        refreshMapViewByGeoPoint
        (
          getGeoByAddress
          (
            mEditText01.getText().toString()
          ),mMapView01,intZoomLevel,true
        );
      }
    }
  });

  /* 放大 */
  mButton02 = (Button)findViewById(R.id.myButton2);
  mButton02.setOnClickListener(new Button.OnClickListener()
  {
    @Override
```

```
    public void onClick(View v)
    {
      // TODO Auto-generated method stub
      intZoomLevel++;
      if(intZoomLevel>mMapView01.getMaxZoomLevel())
      {
        intZoomLevel = mMapView01.getMaxZoomLevel();
      }
      mMapController01.setZoom(intZoomLevel);
    }
  });

  /* 缩小 */
  mButton03 = (Button)findViewById(R.id.myButton3);
  mButton03.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View v)
    {
      // TODO Auto-generated method stub
      intZoomLevel--;
      if(intZoomLevel<1)
      {
        intZoomLevel = 1;
      }
      mMapController01.setZoom(intZoomLevel);
    }
  });

  /* 初次查询地点 */
  refreshMapViewByGeoPoint
  (
    getGeoByAddress
    (
        getResources().getText(R.string.str_default_address).toString()
    ),mMapView01,intZoomLevel,true
  );
}

private GeoPoint getGeoByAddress(String strSearchAddress)
{
  GeoPoint gp = null;
  try
  {
    if(strSearchAddress!="")
    {
      Geocoder mGeocoder01 = new Geocoder(example5.this, Locale.getDefault());
      List<Address> lstAddress = mGeocoder01.getFromLocationName (strSearchAddress, 1);
      if (!lstAddress.isEmpty())
      {
        // Address[addressLines=[0:"U.S PIZZA",1:"15th Main Rd, Phase II, J P
        Nagar",2:"Bengaluru, Karnataka",3:"India"],feature=U.S PIZZA,admin=
```

```
        Karnataka, sub-admin=Bengaluru,locality=Bengaluru,thoroughfare=15th Main
        Rd,postalCode=null, countryCode=IN,countryName=India,hasLatitude=
        true,latitude=117.508933,hasLongitude=true,longitude=73.8042,
        phone=null,url=null,extras=null]
        /*
        for (int i = 0; i < lstAddress.size(); ++i)
        {
          Address adsLocation = lstAddress.get(i);
          Log.i(TAG, "Address found = " + adsLocation.toString());
          //lat = adsLocation.getLatitude();
          //lon = adsLocation.getLongitude();
        }
        */
        Address adsLocation = lstAddress.get(0);
        double geoLatitude = adsLocation.getLatitude()*1E6;
        double geoLongitude = adsLocation.getLongitude()*1E6;
        gp = new GeoPoint((int) geoLatitude, (int) geoLongitude);
      }
      else
      {
        Log.i(TAG, "Address GeoPoint NOT Found.");
      }
    }
  }
  catch (Exception e)
  {
    e.printStackTrace();
  }
  return gp;
}

public static void refreshMapViewByGeoPoint(GeoPoint gp, MapView mv, int
  zoomLevel, boolean bIfSatellite)
{
  try
  {
    mv.displayZoomControls(true);
    /* 取得 MapView 的 MapController */
    MapController mc = mv.getController();
    /* 移至该地理坐标地址 */
    mc.animateTo(gp);
    /* 放大地图层级 */
    mc.setZoom(zoomLevel);
    /* 设置 MapView 的显示选项(卫星、街道)*/
    if(bIfSatellite)
    {
      mv.setSatellite(true);
      mv.setStreetView(true);
    }
    else
    {
      mv.setSatellite(false);
```

```
            }
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    @Override
    protected boolean isRouteDisplayed()
    {
        // TODO Auto-generated method stub
        return false;
    }
}
```

至此，整个演练结束，执行后能够实现地址反查处理，执行效果如图 17-5 所示。



图 17-5　执行效果

17.4　Directions Route 实现路径导航

题　目	目　的	源码路径
演练 4	通过 Directions Route 实现路径导航	“光盘:\daima\17\example6”文件夹

17.4.1　实例分析

在 Android SDK 中，可以调用手机内置地图程序来传递导航坐标来规划路径。在具体实现中，先调用 getLocationProvider()来获取当前 Location，以取得当前所在位置的地理坐标，并通过提供的 EditText Widget 来让用户输入将要前往的地址，通过地址来反复取得目的地地理坐标。

通过两个 GeoPoint 对象，并通过 Intent 方式来调用内置的地图程序。具体实现上，要把握如下两点。

- ❑ 以 Url.parse()传入 Google Map 的路径规划参数方法。
- ❑ 根据手机的移动状态，更改并更新当前 GeoPoint 的方法。

只要设置好了起点 GeoPoint 和终点 GeoPoint，通过重组 Google Map 的 GET 传输参数，便可以传入 Google Map 显示地理坐标。实际上，map.google.com 能够接受的经纬度需要以“经度，纬度”的字符串格式来传递，所以编写了一个 GeoPointToString()来重组 GeoPoint 的经纬度值。

17.4.2　具体实现

本实例的主文件是 example6.java，下面是具体实现的流程。

(1) 创建 MapView 对象，通过创建 LocationManager 对象取得系统 Location 服务，然后设置 MapView 的显示选项(卫星、街道)。具体实现代码如下所示：

```
protected void onCreate(Bundle icicle)
{
  super.onCreate(icicle);
  setContentView(R.layout.main);
  mTextView01 = (TextView)findViewById(R.id.myTextView1);
  mEditText01 = (EditText)findViewById(R.id.myEditText1);
  mEditText01.setText
  (
    getResources().getText
    (R.string.str_default_address).toString()
  );

  /* 创建 MapView 对象 */
  mMapView01 = (MapView)findViewById(R.id.myMapView1);
  mMapController01 = mMapView01.getController();

  // 设置 MapView 的显示选项(卫星、街道)
  mMapView01.setSatellite(true);
  mMapView01.setStreetView(true);

  // 放大的层级
  intZoomLevel = 15;
  mMapController01.setZoom(intZoomLevel);

  /* 创建 LocationManager 对象取得系统 LOCATION 服务 */
  mLocationManager01 =
  (LocationManager)getSystemService(Context.LOCATION_SERVICE);

  /*
   * 自定义函数，访问 Location Provider，
   * 并将之存储在 strLocationProvider 当中
   */
  getLocationProvider();
  /* 传入 Location 对象，显示于 MapView */
  fromGeoPoint = getGeoByLocation(mLocation01);
```

```
refreshMapViewByGeoPoint(fromGeoPoint,
                mMapView01, intZoomLevel);
/* 创建 LocationManager 对象，监听
 * Location 更改事件，更新 MapView*/
mLocationManager01.requestLocationUpdates
(strLocationProvider, 2000, 10, mLocationListener01);
```

(2) 定义 toGeoPoint 获取 User 要前往地址的 GeoPoint 对象，传入路径规划所需要的地标地址。具体实现代码如下所示：

```
mButton01 = (Button)findViewById(R.id.myButton1);
mButton01.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    // TODO Auto-generated method stub
    if(mEditText01.getText().toString()!="")
    {
      /* 取得 User 要前往地址的 GeoPoint 对象 */
      toGeoPoint =
      getGeoByAddress(mEditText01.getText().toString());
      /* 路径规划 Intent */
      Intent intent = new Intent();
      intent.setAction(android.content.Intent.ACTION_VIEW);
      /* 传入路径规划所需要的地标地址 */
      intent.setData
      (
        Uri.parse("http://maps.google.com/maps?f=d&saddr="+
        GeoPointToString(fromGeoPoint)+
        "&daddr="+GeoPointToString(toGeoPoint)+
        "&hl=cn" +
        "")
      );
      startActivity(intent);
    }
  }
});
```

(3) 定义 mButton02 按钮处理事件，实现地图放大处理。定义 mButton03 按钮处理事件，实现地图缩小处理。具体代码如下所示：

```
/* 放大地图 */
mButton02 = (Button)findViewById(R.id.myButton2);
mButton02.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    // TODO Auto-generated method stub
    intZoomLevel++;
    if(intZoomLevel>mMapView01.getMaxZoomLevel())
    {
```

```
        intZoomLevel = mMapView01.getMaxZoomLevel();
      }
      mMapController01.setZoom(intZoomLevel);
    }
  });

  /* 缩小地图 */
  mButton03 = (Button)findViewById(R.id.myButton3);
  mButton03.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View v)
    {
      // TODO Auto-generated method stub
      intZoomLevel--;
      if(intZoomLevel<1)
      {
        intZoomLevel = 1;
      }
      mMapController01.setZoom(intZoomLevel);
    }
  });
}
```

(4) 捕捉手机 GPS 坐标更新时的事件，当手机收到位置更改时，将 location 传入 getMyLocation。具体代码如下：

```
/* 捕捉当手机 GPS 坐标更新时的事件 */
public final LocationListener mLocationListener01 =
new LocationListener()
{
  @Override
  public void onLocationChanged(Location location)
  {
    // TODO Auto-generated method stub

    /* 当手机收到位置更改时，将 location 传入 getMyLocation */
    mLocation01 = location;
    fromGeoPoint = getGeoByLocation(location);
    refreshMapViewByGeoPoint(fromGeoPoint,
          mMapView01, intZoomLevel);
  }

  @Override
  public void onProviderDisabled(String provider)
  {
    // TODO Auto-generated method stub
    mLocation01 = null;
  }

  @Override
  public void onProviderEnabled(String provider)
```

```
  {
    // TODO Auto-generated method stub

  }

  @Override
  public void onStatusChanged(String provider,
            int status, Bundle extras)
  {
    // TODO Auto-generated method stub

  }
};
```

(5) 定义方法 getGeoByLocation(Location location)，当传入 Location 对象时取回 GeoPoint 对象。具体代码如下：

```
/* 传入 Location 对象，取回其 GeoPoint 对象 */
private GeoPoint getGeoByLocation(Location location)
{
  GeoPoint gp = null;
  try
  {
    /* 当 Location 存在 */
    if (location != null)
    {
      double geoLatitude = location.getLatitude()*1E6;
      double geoLongitude = location.getLongitude()*1E6;
      gp = new GeoPoint((int) geoLatitude, (int) geoLongitude);
    }
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
  return gp;
}
```

(6) 定义方法 getGeoByAddress(String strSearchAddress)，当输入地址时取得 GeoPoint 对象。具体代码如下：

```
/* 输入地址，取得其 GeoPoint 对象 */
private GeoPoint getGeoByAddress(String strSearchAddress)
{
  GeoPoint gp = null;
  try
  {
    if(strSearchAddress!="")
    {
      Geocoder mGeocoder01 = new Geocoder
      (example6.this, Locale.getDefault());
      List<Address> lstAddress = mGeocoder01.getFromLocationName
```

```
                        (strSearchAddress, 1);
       if (!lstAddress.isEmpty())
       {
         Address adsLocation = lstAddress.get(0);
         double geoLatitude = adsLocation.getLatitude()*1E6;
         double geoLongitude = adsLocation.getLongitude()*1E6;
         gp = new GeoPoint((int) geoLatitude, (int) geoLongitude);
       }
     }
   }
   catch (Exception e)
   {
     e.printStackTrace();
   }
   return gp;
 }
```

(7) 定义 refreshMapViewByGeoPoint 和 refreshMapViewByCode，分别传入 geoPoint 更新 MapView 里的 Google Map 和传入经纬度更新 MapView 里的 Google Map。具体代码如下：

```
/* 传入 geoPoint 更新 MapView 里的 Google Map */
public static void refreshMapViewByGeoPoint
(GeoPoint gp, MapView mapview, int zoomLevel)
{
  try
  {
    mapview.displayZoomControls(true);
    MapController myMC = mapview.getController();
    myMC.animateTo(gp);
    myMC.setZoom(zoomLevel);
    mapview.setSatellite(false);
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}

/* 传入经纬度更新 MapView 里的 Google Map */
public static void refreshMapViewByCode
(double latitude, double longitude,
    MapView mapview, int zoomLevel)
{
  try
  {
    GeoPoint p = new GeoPoint((int) latitude, (int) longitude);
    mapview.displayZoomControls(true);
    MapController myMC = mapview.getController();
    myMC.animateTo(p);
    myMC.setZoom(zoomLevel);
    mapview.setSatellite(false);
  }
```

```
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }
```

(8) 定义方法 GeoPointToString(GeoPoint gp)，将 GeoPoint 里的经纬度以 String,String 返回。具体代码如下：

```
/* 将 GeoPoint 里的经纬度以 String, String 返回 */
private String GeoPointToString(GeoPoint gp)
{
  String strReturn="";
  try
  {
    /* 当 Location 存在 */
    if (gp != null)
    {
      double geoLatitude = (int)gp.getLatitudeE6()/1E6;
      double geoLongitude = (int)gp.getLongitudeE6()/1E6;
      strReturn = String.valueOf(geoLatitude)+","+
        String.valueOf(geoLongitude);
    }
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
  return strReturn;
}
```

(9) 定义方法 getLocationProvider()，用于获取 LocationProvider。具体代码如下：

```
/* 取得 LocationProvider */
public void getLocationProvider()
{
  try
  {
    Criteria mCriteria01 = new Criteria();
    mCriteria01.setAccuracy(Criteria.ACCURACY_FINE);
    mCriteria01.setAltitudeRequired(false);
    mCriteria01.setBearingRequired(false);
    mCriteria01.setCostAllowed(true);
    mCriteria01.setPowerRequirement(Criteria.POWER_LOW);
    strLocationProvider =
    mLocationManager01.getBestProvider(mCriteria01, true);

    mLocation01 = mLocationManager01.getLastKnownLocation
    (strLocationProvider);
  }
  catch(Exception e)
  {
    mTextView01.setText(e.toString());
```

```
      e.printStackTrace();
    }
  }

  @Override
  protected boolean isRouteDisplayed()
  {
    // TODO Auto-generated method stub
    return false;
  }
}
```

至此，整个演练结束，执行后的效果如图 17-6 所示；单击“开始规划路径”按钮后弹出选择对话框，如图 17-7 所示。

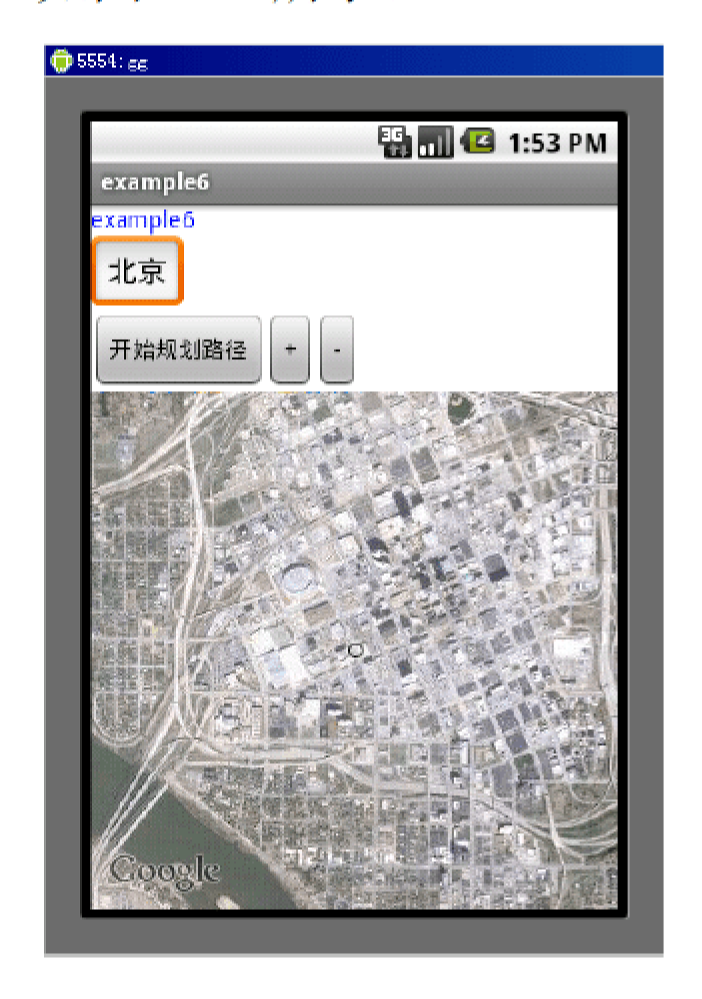


图 17-6 执行效果

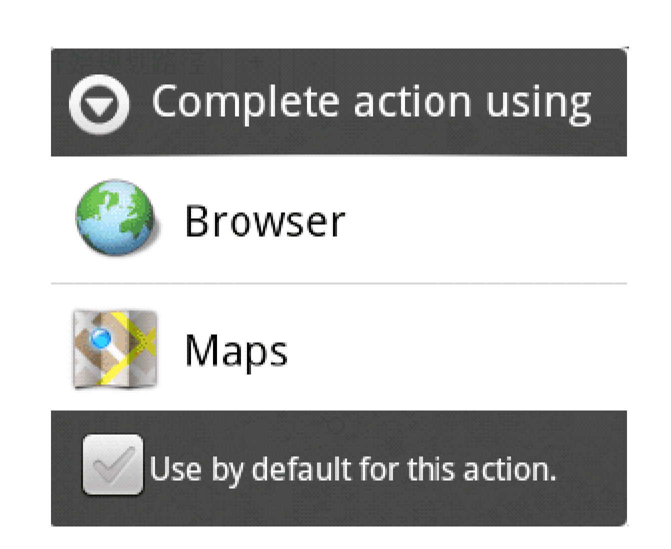


图 17-7 选择对话框

在图 17-7 中选择 Maps 后弹出规划界面，如图 17-8 所示；在图 17-8 所示的界面中，在第一个文本框中设置出发位置，例如“beijing”，在第二个文本框中设置目的地位置，例如“tianjin”，单击汽车前往标志按钮，如图 17-9 所示。

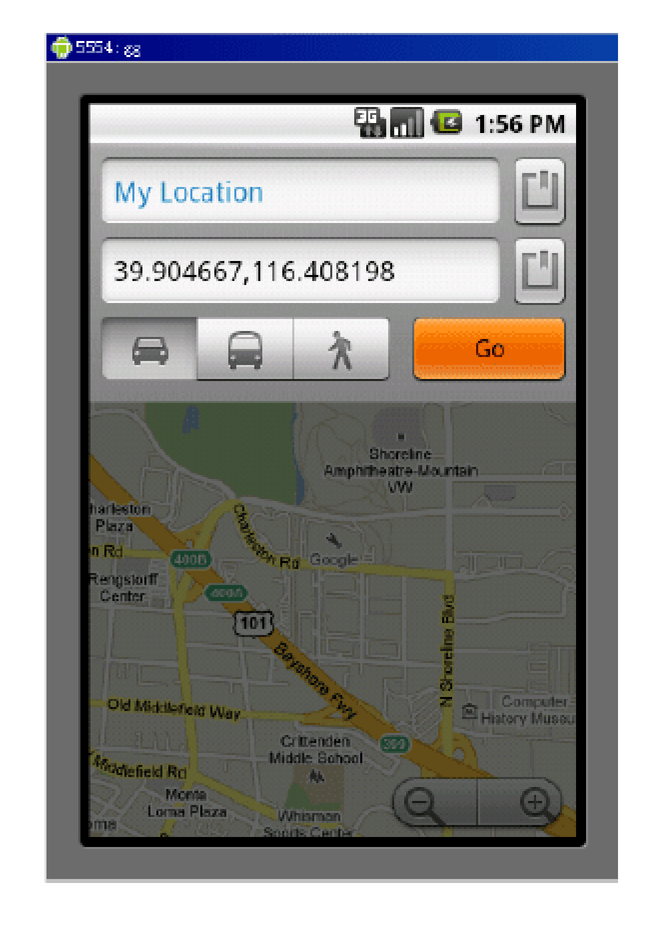


图 17-8 规划界面

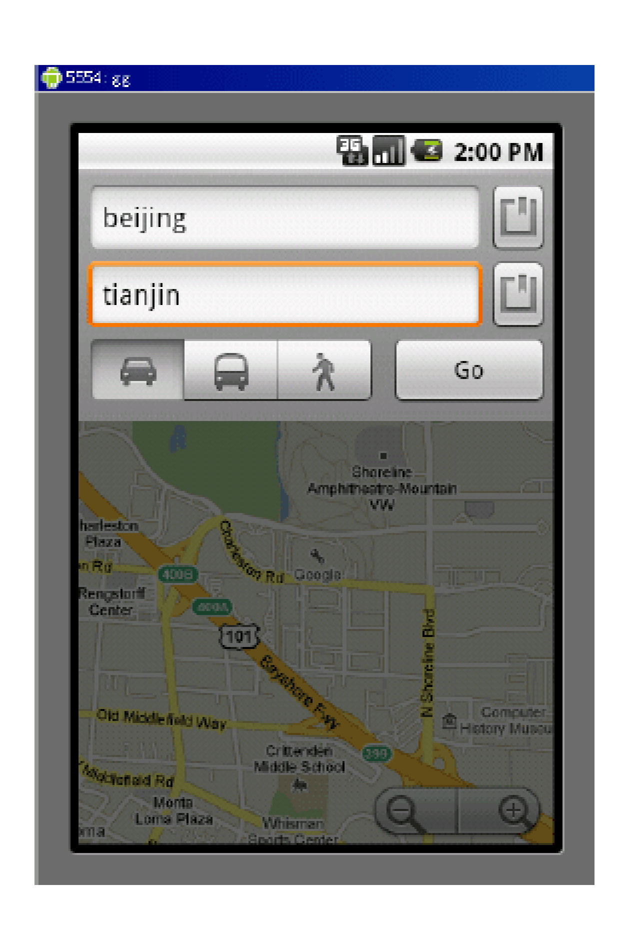


图 17-9 设置出发地和目的地

然后，单击 Go 按钮后，系统将实现从“beijing”出发，目的地到“tianjin”的线路规划，产生线路规划图，最终的执行界面如图 17-10 所示。当单击图 17-10 中的 Show on map 按钮后，将会在地图中显示行走线路，线路规划如图 17-11 所示。

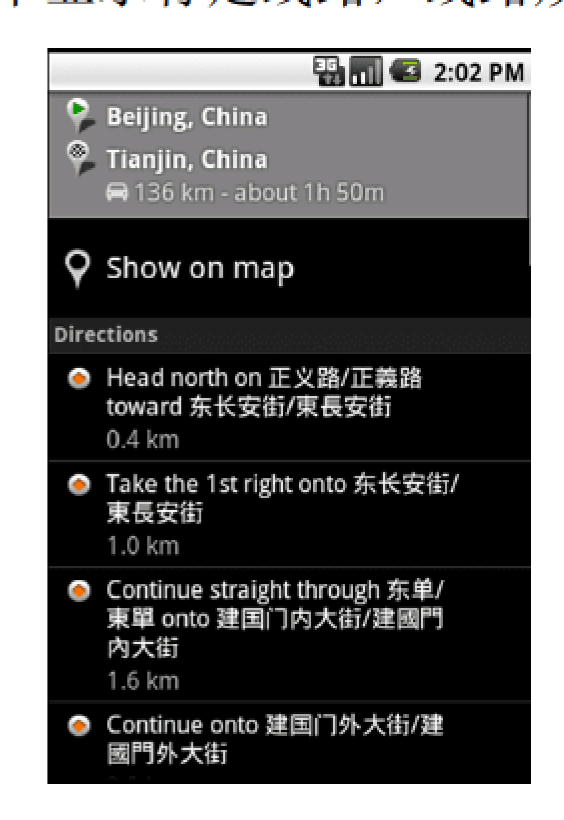




图 17-10　生成的线路规划

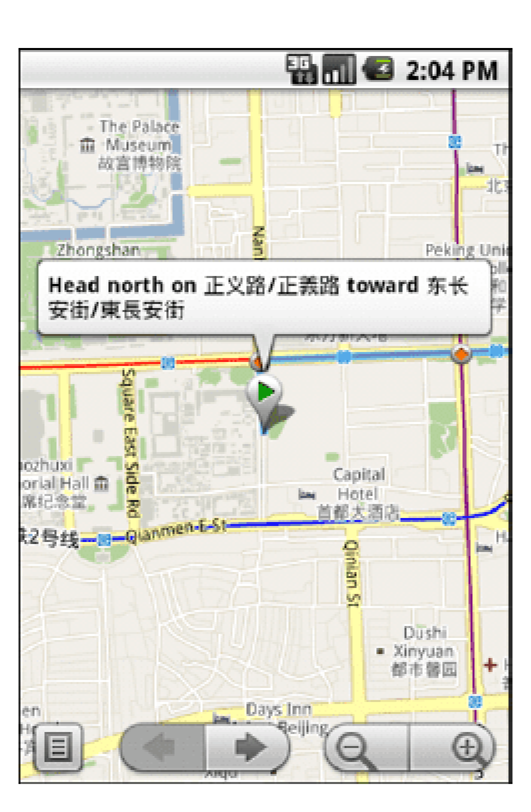


图 17-11　地图中的线路规划

注意：出发地和目的地不能属于两个不同的国家，否则将会产生错误提示。

17.5　LocationListener 和 MapView 实时更新

题　目	目　的	源码路径
演练 5	实现 GPS 实时更新处理	“光盘:\daima\17\example7”文件夹

17.5.1　GPS 的使用

在现实应用中，GPS 的使用范围越来越广。但是，每一个位置和路况都不是固定不变的，这就要求系统能够根据各种实时变化而变化，不能误导人们的使用。一方面，在 Android SDK 中，支持手机 GPS 定位事件处理；另一方面，虽然在 Android 手机中内置了 Google map，但是不具备随着手机的移动而更新功能，这就要求我们程序员使用 Android SDK 来开发及时更新程序。在本实例中，将插入一个 TextView 和 MapView，当手机移动时，会触发内置的 GPS 定位坐标的改变事件。只要程序发现地理位置变化，便实时更新 MapView 里的 Google Map，并反查地理坐标系统的信息。

17.5.2　具体实现

本实例的主文件是 example7.java，下面开始介绍其实现流程。

(1)　创建 MapView 对象 mMapView01，然后创建 LocationManager 对象 mLocationManager01 来获取系统 Location 服务。具体代码如下：

```
super.onCreate(icicle);
setContentView(R.layout.main);
```

```
  mTextView01 = (TextView)findViewById(R.id.myTextView1);
  /* 创建 MapView 对象 */
  mMapView01 = (MapView)findViewById(R.id.myMapView1);

  /* 创建 LocationManager 对象取得系统 LOCATION 服务 */
  mLocationManager01 =
  (LocationManager)getSystemService(Context.LOCATION_SERVICE);
```

(2) 通过 getLocationProvider 取得当前 Location，然后创建 LocationManager 对象用于监听 Location 更改事件，并更新 MapView。具体代码如下：

```
  /* 第一次运行向 Location Provider 取得 Location */
  mLocation01 = getLocationProvider(mLocationManager01);

  if(mLocation01!=null)
  {
    processLocationUpdated(mLocation01);
  }
  else
  {
    mTextView01.setText
    (
      getResources().getText(R.string.str_err_location).toString()
    );
  }
  /* 创建 LocationManager 对象，监听 Location 更改事件，更新 MapView */
  mLocationManager01.requestLocationUpdates
  (strLocationProvider, 2000, 10, mLocationListener01);
}
```

(3) 通过 LocationListener()监听定位信息，当手机收到位置更改时，将 location 传入取得地理坐标。具体代码如下：

```
public final LocationListener
mLocationListener01 = new LocationListener()
{
  @Override
  public void onLocationChanged(Location location)
  {
    // TODO Auto-generated method stub
    /* 当手机收到位置更改时，将 location 传入取得地理坐标 */
    processLocationUpdated(location);
  }
  @Override
  public void onProviderDisabled(String provider)
  {
    // TODO Auto-generated method stub
    /* 当 Provider 已离开服务范围时 */
  }
  @Override
  public void onProviderEnabled(String provider)
  {
    // TODO Auto-generated method stub
```

```
    }
    @Override
    public void onStatusChanged
    (String provider, int status, Bundle extras)
    {
      // TODO Auto-generated method stub

    }
  };
```

(4) 定义方法 getAddressbyGeoPoint(GeoPoint gp)，用于获取定位地址的信息。先创建 Geocoder 对象并取出地理坐标经纬度，然后判断地址是否为多行，最后将撷取到的地址组合放在 StringBuilder 对象中输出。具体代码如下：

```
public String getAddressbyGeoPoint(GeoPoint gp)
{
  String strReturn = "";
  try
  {
    /* 当 GeoPoint 不等于 null */
    if (gp != null)
    {
      /* 创建 Geocoder 对象 */
      Geocoder gc = new Geocoder
      (example7.this, Locale.getDefault());

      /* 取出地理坐标经纬度 */
      double geoLatitude = (int)gp.getLatitudeE6()/1E6;
      double geoLongitude = (int)gp.getLongitudeE6()/1E6;

      /* 自经纬度取得地址(可能有多行地址) */
      List<Address> lstAddress =
      gc.getFromLocation(geoLatitude, geoLongitude, 1);
      StringBuilder sb = new StringBuilder();

      /* 判断地址是否为多行 */
      if (lstAddress.size() > 0)
      {
        Address adsLocation = lstAddress.get(0);
        for(int i=0;i<adsLocation.getMaxAddressLineIndex();i++)
        {
          sb.append(adsLocation.getAddressLine(i)).append("\n");
        }
        sb.append(adsLocation.getLocality()).append("\n");
        sb.append(adsLocation.getPostalCode()).append("\n");
        sb.append(adsLocation.getCountryName());
      }

      /* 将撷取到的地址组合放在 StringBuilder 对象中输出用 */
      strReturn = sb.toString();
    }
```

```
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
    return strReturn;
  }

  public Location getLocationProvider(LocationManager lm)
  {
    Location retLocation = null;
    try
    {
      Criteria mCriteria01 = new Criteria();
      mCriteria01.setAccuracy(Criteria.ACCURACY_FINE);
      mCriteria01.setAltitudeRequired(false);
      mCriteria01.setBearingRequired(false);
      mCriteria01.setCostAllowed(true);
      mCriteria01.setPowerRequirement(Criteria.POWER_LOW);
      strLocationProvider = lm.getBestProvider(mCriteria01, true);
      retLocation = lm.getLastKnownLocation(strLocationProvider);
    }
    catch(Exception e)
    {
      mTextView01.setText(e.toString());
      e.printStackTrace();
    }
    return retLocation;
  }
  private GeoPoint getGeoByLocation(Location location)
  {
    GeoPoint gp = null;
    try
    {
      /* 当Location存在 */
      if (location != null)
      {
        double geoLatitude = location.getLatitude()*1E6;
        double geoLongitude = location.getLongitude()*1E6;
        gp = new GeoPoint((int) geoLatitude, (int) geoLongitude);
      }
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
    return gp;
  }
  public static void refreshMapViewByGeoPoint
  (GeoPoint gp, MapView mv, int zoomLevel, boolean bIfSatellite)
```

```
{
  try
  {
    mv.displayZoomControls(true);
    /* 取得 MapView 的 MapController */
    MapController mc = mv.getController();
    /* 移至该地理坐标地址 */
    mc.animateTo(gp);

    /* 放大地图层级 */
    mc.setZoom(zoomLevel);

    /* 设置 MapView 的显示选项(卫星、街道)*/
    if(bIfSatellite)
    {
      mv.setSatellite(true);
      mv.setStreetView(true);
    }
    else
    {
      mv.setSatellite(false);
    }
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}
```

(5) 定义方法 processLocationUpdated(Location location)，当手机收到位置更改时，将 location 传入 GeoPoint 及 MapView 中。具体代码如下：

```
/* 当手机收到位置更改时，将 location 传入 GeoPoint 及 MapView */
private void processLocationUpdated(Location location)
{
  /* 传入 Location 对象，取得 GeoPoint 地理坐标 */
  currentGeoPoint = getGeoByLocation(location);
  /* 更新 MapView 显示 Google Map */
  refreshMapViewByGeoPoint
  (currentGeoPoint, mMapView01, intZoomLevel, true);
  mTextView01.setText
  (
    getResources().getText(R.string.str_my_location).toString()+
    "\n"+getAddressbyGeoPoint(currentGeoPoint)
  );
}
@Override
protected boolean isRouteDisplayed()
{
  // TODO Auto-generated method stub
  return false;
```

```
    }
}
```

至此，整个演练结束，执行后将显示当前位置的定位信息，执行效果如图 17-12 所示并能实现及时更新。

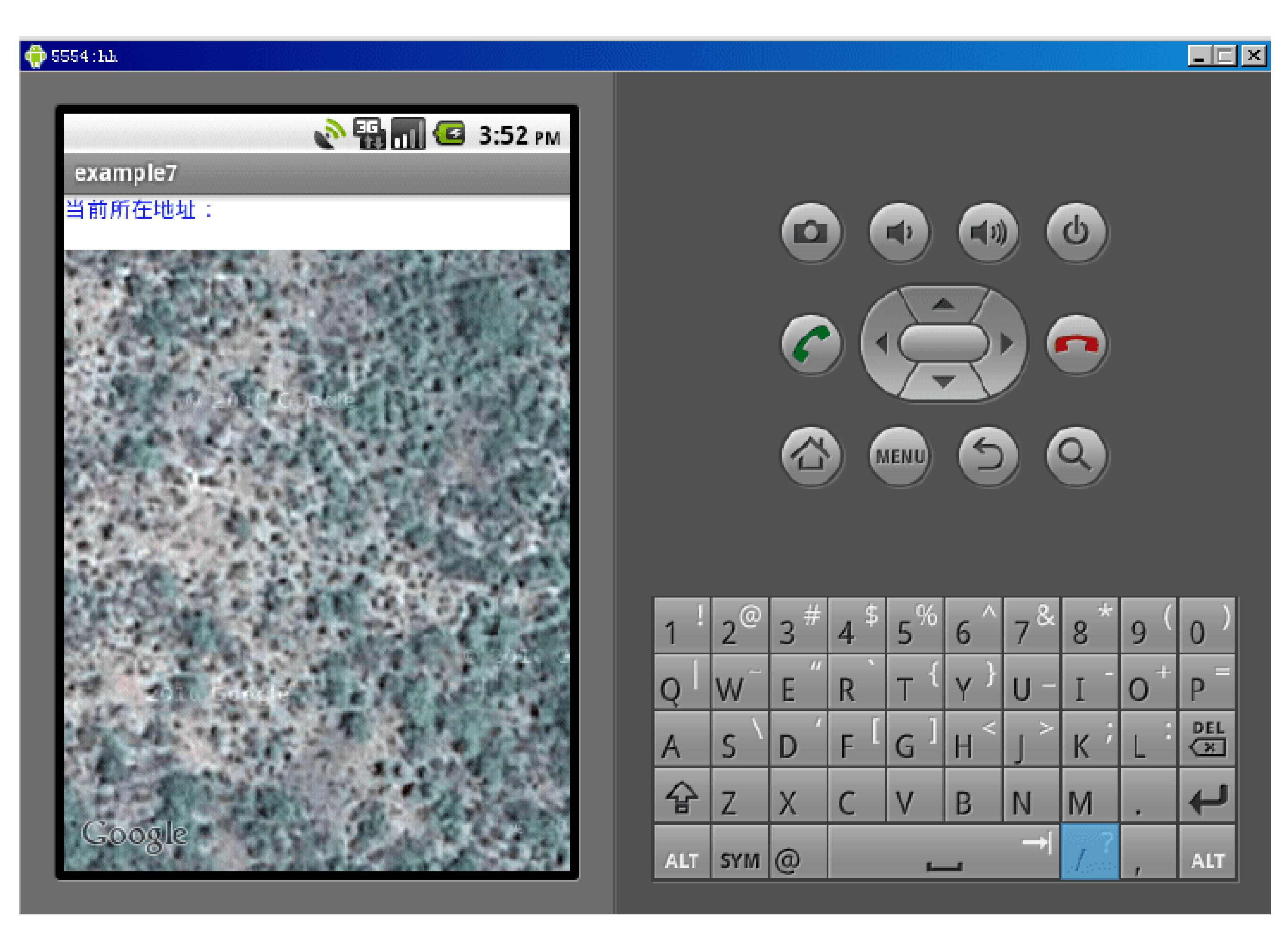


图 17-12　执行效果

17.6　Google Translate API 翻译

题　目	目　的	源码路径
演练 6	通过 Google Translate API 实现翻译处理	“光盘:\daima\17\example8”文件夹

17.6.1　Google Translate API

在 Android 系统中是没有手机翻译功能的，但是 Android 官方为其提供了 API 支持，通过调用 Google Translate API 即可实现翻译功能。Google 在线翻译功能十分强大，现在 Google 已经开放了 Ajax 的 API。使用 Ajax 语言的 API，可以仅使用 JavaScript 来翻译和检测网页中文本块的语言。此外，也可以在网页的任何文本字段或文本区域启用音译，例如，如果您已音译为北印度语，则该 API 会允许用户使用英语按照发音拼出北印度语单词，并以北印度语脚本显示出来。

google-api-translate-java 是 Java 语言对 Google 翻译引擎的封装类库，具体使用方法如下：

```
import com.google.api.translate.Language;
import com.google.api.translate.Translate;
public class Main {
  public static void main(String[] args) {
    try {
      String translatedText =
          Translate.translate("Salut  le  monde",  Language.FRENCH,  Language.
ENGLISH);
```

```
        System.out.println(translatedText);
      } catch (Exception ex) {
        ex.printStackTrace();
      }
    }
}
```

17.6.2 具体实现

本实例的主文件是 example8.java，下面是其具体实现流程。

(1) 定义 WebSettings 对象 webSettings，用于获取 WebSettings。具体代码如下：

```
    myEditText1 = (EditText) findViewById(R.id.myEditText1);
    myWebView1 = (WebView) findViewById(R.id.myWebView1);
    /* 取得 WebSettings */
    WebSettings webSettings = myWebView1.getSettings();
    /* 设置可运行 JavaScript */
    webSettings.setJavaScriptEnabled(true);
    webSettings.setSaveFormData(false);
    webSettings.setSavePassword(false);
    webSettings.setSupportZoom(false);
    myWebView1.setWebChromeClient(new MyWebChromeClient());
    /* 设置给 html 调用的对象及名称 */
    myWebView1.addJavascriptInterface(new runJavaScript(), "irdc");
    /* 将 assets/google_translate.html 载入 */
    String url = "file:///android asset/google translate.html";
    myWebView1.loadUrl(url);
  }
```

(2) 定义 runJavaScript，用于调用 google_translate.html 里的 JavaScript 以显示结果。具体代码如下：

```
  final class runJavaScript
  {
    public void runOnAndroidJavaScript()
    {
      mHandler01.post(new Runnable()
      {
        public void run()
        {
          if (myEditText1.getText().toString() != "")
          {
            /* 调用 google_translate.html 里的 javascript */
            myWebView1.loadUrl("javascript:translate('"
                + myEditText1.getText().toString() + "')");
          }
        }
      });
    }
  }
```

(3) 定义 onJsAlert，用于捕捉网页里的 alert javascript 作为.js 调试之用，并输出至 LogCat。

具体代码如下：

```
/**
 * 捕捉网页里的 alert javascript 作为.js 调试之用，并输出至 LogCat
 */
final class MyWebChromeClient extends WebChromeClient
{
  @Override
  public boolean onJsAlert(WebView view, String url,
      String message, JsResult result)
  {
    // TODO Auto-generated method stub
    Log.d(LOG_TAG, message);
    // result.confirm();
    return super.onJsAlert(view, url, message, result);
  }
 }
}
```

至此，整个演练结束，执行后的效果如图 17-13 所示；当输入英文字符并单击“中文”链接后，将分别弹出两个对话框，分别如图 17-14 和图 17-15 所示；依次单击 OK 按钮即可实现翻译处理，显示的翻译结果如图 17-16 所示。

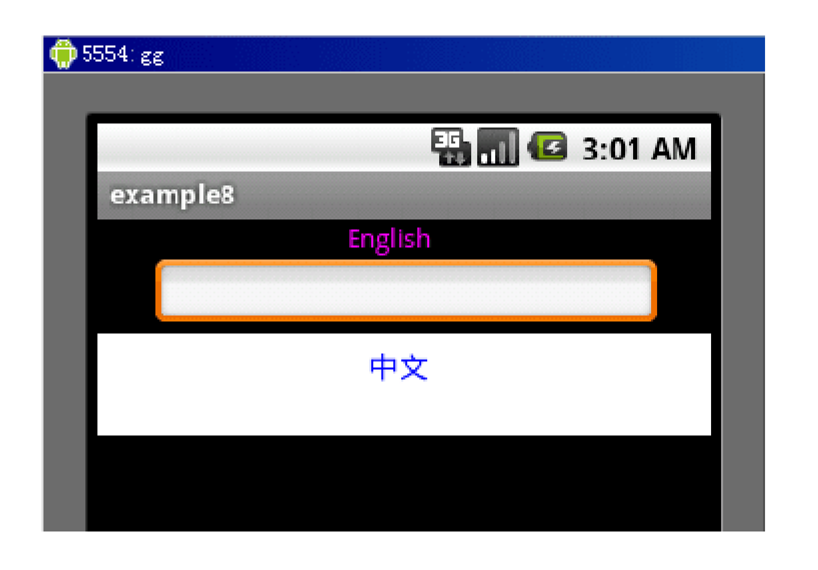


图 17-13　初始效果

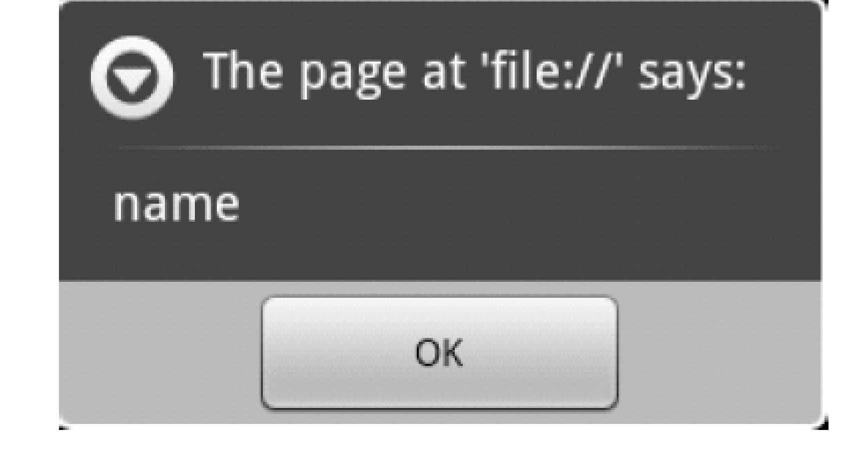


图 17-14　单击 OK 按钮

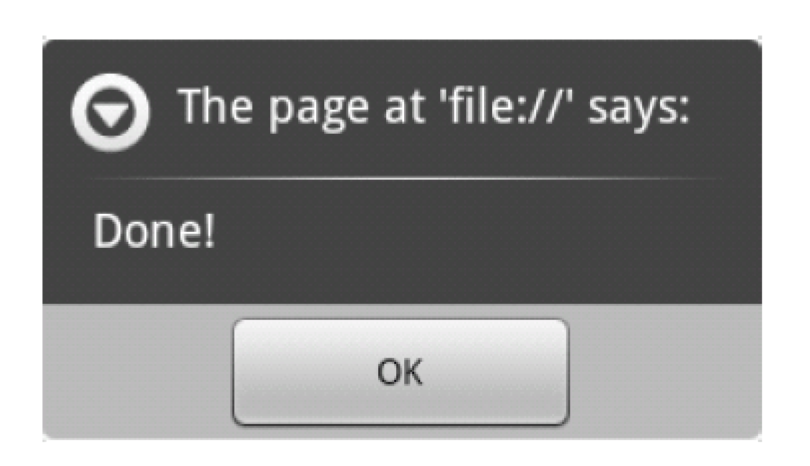


图 17-15　单击 OK 按钮



图 17-16　翻译结果

17.7　画图并计算距离

题　目	目　的	源码路径
演练 7	在地图上画图并计算距离	“光盘:\daima\17\example9”文件夹

17.7.1 绘制地图

除了导航之外，还可以在地图上绘制线路、计算距离，实现一个完整的导航功能，Google Web Toolkit(GWT)引入了 JavaScript overlay 类型以简化将整个 JavaScript 对象家族集成到 GWT 项目的过程。该技术有很多优势，如利用 Java IDE 的代码完成和重构能力，甚至当你在编写无类型的 JavaScript 对象时也可以充分利用这一优势。

通过 overlay 类可以在地图上绘制图形或添加图片，在使用前需要引用，具体格式如下：

```
import com.google.android.maps.Overlay;
```

在本实例中，设置了一个继承 com.google.android.maps.Overlay 的类 MyOverLay，并对方法 onDraw()进行了重写，这样实现了在 MapView 中添加轨迹的效果。

17.7.2 具体实现

本实例的主文件是 example9.java 和 OverLay.java，下面是具体的实现流程。

1. 文件 example9.java

文件 example9.java 的功能是通过自定义的 MyOveryLay 类在 MapView 中画上标记。具体实现流程如下所示。

(1) 设置默认的放大层级为 17 级，对 Provider 初始化处理并分别获取 Provider 与 Location，取得当前位置。具体代码如下：

```
/* 设置默认的放大层级 */
zoomLevel = 17;
mMapController.setZoom(zoomLevel);

/* Provider 初始化 */
mLocationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
/* 取得 Provider 与 Location */
getLocationPrivider();
if(mLocation!=null)
{
  /* 取得目前的经纬度 */
  gp1=getGeoByLocation(mLocation);
  gp2=gp1;
  /* 将 MapView 的中点移至目前位置 */
  refreshMapView();
  /* 设置事件的 Listener */
  mLocationManager.requestLocationUpdates(mLocationPrivider,
      2000, 10, mLocationListener);
}
else
{
  new AlertDialog.Builder(example17.this).setTitle("系统信息")
```

```
        .setMessage(getResources().getString(R.string.str_message))
        .setNegativeButton("确定",new DialogInterface.OnClickListener()
         {
           public void onClick(DialogInterface dialog, int which)
           {
             example17.this.finish();
           }
         })
         .show();
    }
```

(2) 定义方法 mButton01.setOnClickListener，用于响应单击“开始记录”按钮后的处理事件。具体代码如下：

```
/* 开始记录的 Button */
mButton01.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    gp1=gp2;
    /* 清除 Overlay */
    resetOverlay();
    /* 画起点 */
    setStartPoint();
    /* 更新 MapView */
    refreshMapView();
    /* 重设移动距离为 0，并更新 TextView */
    distance=0;
    mTextView.setText("移动距离：0M");
    /* 启动画路线的机制 */
    _run=true;
  }
});
```

(3) 定义方法 mButton02.setOnClickListener，用于响应单击“结束记录”按钮后的处理事件。具体代码如下：

```
/* 结束记录的 Button */
mButton02.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    /* 画终点 */
    setEndPoint();
    /* 更新 MapView */
    refreshMapView();
    /* 终止画路线的机制 */
    _run=false;
  }
});
```

(4) 定义方法 mButton03.setOnClickListener，用于响应单击“缩小地图”按钮后的处理事件。具体代码如下：

```
/* 缩小地图的 Button */
mButton03.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    zoomLevel--;
    if(zoomLevel<1)
    {
      zoomLevel = 1;
    }
    mMapController.setZoom(zoomLevel);
  }
});
```

(5) 定义方法 mButton04.setOnClickListener，用于响应单击“放大地图”按钮后的处理事件。具体代码如下：

```
  /* 放大地图的 Button */
  mButton04.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View v)
    {
      zoomLevel++;
      if(zoomLevel>mMapView.getMaxZoomLevel())
      {
        zoomLevel = mMapView.getMaxZoomLevel();
      }
      mMapController.setZoom(zoomLevel);
    }
  });
}
```

(6) 定义方法 onLocationChanged(Location location)，用于监听当前位置的变化，如果变化则记下轨迹线路。具体代码如下：

```
/* MapView 的 Listener */
public final LocationListener mLocationListener =
  new LocationListener()
{
  @Override
  public void onLocationChanged(Location location)
  {
    /* 如果记录进行中，就画路线并更新移动距离 */
    if(_run)
    {
      /* 记下移动后的位置 */
      gp2=getGeoByLocation(location);
```

```
      /* 画路线 */
      setRoute();
      /* 更新 MapView */
      refreshMapView();
      /* 取得移动距离 */
      distance+=GetDistance(gp1,gp2);
      mTextView.setText("移动距离："+format(distance)+"M");

      gp1=gp2;
    }
  }
  @Override
  public void onProviderDisabled(String provider)
  {
  }
  @Override
  public void onProviderEnabled(String provider)
  {
  }
  @Override
  public void onStatusChanged(String provider,int status, Bundle extras)
  {
  }
};
```

(7)　定义方法 getGeoByLocation(Location location)，用于取得 GeoPoint 的方法，具体代码如下：

```
/* 取得 GeoPoint 的方法 */
private GeoPoint getGeoByLocation(Location location)
{
  GeoPoint gp = null;
  try
  {
    if (location != null)
    {
      double geoLatitude = location.getLatitude()*1E6;
      double geoLongitude = location.getLongitude()*1E6;
      gp = new GeoPoint((int) geoLatitude, (int) geoLongitude);
    }
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
  return gp;
}
```

(8)　定义方法 getLocationPrivider()，用于获取 LocationProvider。具体代码如下：

```
/* 取得 LocationProvider */
public void getLocationPrivider()
{
```

```
    Criteria mCriteria01 = new Criteria();
    mCriteria01.setAccuracy(Criteria.ACCURACY_FINE);
    mCriteria01.setAltitudeRequired(false);
    mCriteria01.setBearingRequired(false);
    mCriteria01.setCostAllowed(true);
    mCriteria01.setPowerRequirement(Criteria.POWER_LOW);
    mLocationPrivider = mLocationManager .getBestProvider(mCriteria01, true);
    mLocation = mLocationManager.getLastKnownLocation(mLocationPrivider);
  }
```

(9) 分别设置起点方法 setStartPoint()、路线方法 setRoute()、终点方法 setEndPoint()、重设 Overlay 方法 resetOverlay()、更新 MapView 方法 refreshMapview()。具体代码如下：

```
/* 设置起点的方法 */
private void setStartPoint()
{
  int mode=1;
  OverLay mOverlay = new OverLay(gp1,gp2,mode);
  List<Overlay> overlays = mMapView.getOverlays();
  overlays.add(mOverlay);
}
/* 设置路线的方法 */
private void setRoute()
{
  int mode=2;
  OverLay mOverlay = new OverLay(gp1,gp2,mode);
  List<Overlay> overlays = mMapView.getOverlays();
  overlays.add(mOverlay);
}
/* 设置终点的方法 */
private void setEndPoint()
{
  int mode=3;
  OverLay mOverlay = new OverLay(gp1,gp2,mode);
  List<Overlay> overlays = mMapView.getOverlays();
  overlays.add(mOverlay);
}
/* 重设 Overlay 的方法 */
private void resetOverlay()
{
  List<Overlay> overlays = mMapView.getOverlays();
  overlays.clear();
}
/* 更新 MapView 的方法 */
public void refreshMapView()
{
  mMapView.displayZoomControls(true);
  MapController myMC = mMapView.getController();
  myMC.animateTo(gp2);
  myMC.setZoom(zoomLevel);
  mMapView.setSatellite(false);
}
```

(10) 定义方法 GetDistance(GeoPoint gp1,GeoPoint gp2)，用于获取两点间的距离，并通过方法 format(double num)处理移动的距离。具体代码如下：

```
  /* 取得两点间的距离的方法 */
  public double GetDistance(GeoPoint gp1,GeoPoint gp2)
  {
    double Lat1r = ConvertDegreeToRadians(gp1.getLatitudeE6()/1E6);
    double Lat2r = ConvertDegreeToRadians(gp2.getLatitudeE6()/1E6);
    double Long1r= ConvertDegreeToRadians(gp1.getLongitudeE6()/1E6);
    double Long2r= ConvertDegreeToRadians(gp2.getLongitudeE6()/1E6);
    /* 地球半径(KM) */
    double R = 6371;
    double d = Math.acos(Math.sin(Lat1r)*Math.sin(Lat2r)+
             Math.cos(Lat1r)*Math.cos(Lat2r)*
             Math.cos(Long2r-Long1r))*R;
    return d*1000;
  }

  private double ConvertDegreeToRadians(double degrees)
  {
    return (Math.PI/180)*degrees;
  }
  /* format 移动距离的方法 */
  public String format(double num)
  {
    NumberFormat formatter = new DecimalFormat("###");
    String s=formatter.format(num);
    return s;
  }
  @Override
  protected boolean isRouteDisplayed()
  {
    return false;
  }
}
```

2. 文件 OverLay.java

文件 OverLay.java 的功能是在 MapView 上绘制图形，继承自 Overlay 类并以 getProjection() 获取 Projection 对象，再以 projection.toPixels(gp1, point)将 getProjection()转换成 Point，再利用 Point 对象的对应位置来绘制图形。具体代码如下：

```
public class OverLay extends Overlay
{
  private GeoPoint gp1;
  private GeoPoint gp2;
  private int mRadius=6;
  private int mode=0;
  /* 构造器，传入起点与终点的 GeoPoint 与 mode */
  public OverLay(GeoPoint gp1,GeoPoint gp2,int mode)
  {
    this.gp1 = gp1;
```

```
    this.gp2 = gp2;
    this.mode = mode;
  }
  @Override
  public boolean draw
  (Canvas canvas, MapView mapView, boolean shadow, long when)
  {
    Projection projection = mapView.getProjection();
    if (shadow == false)
    {
      /* 设置笔刷 */
      Paint paint = new Paint();
      paint.setAntiAlias(true);
      paint.setColor(Color.BLUE);

      Point point = new Point();
      projection.toPixels(gp1, point);
      /* mode=1：创建起点 */
      if(mode==1)
      {
        /* 定义 RectF 对象 */
        RectF oval=new RectF(point.x - mRadius, point.y - mRadius,
                             point.x + mRadius, point.y + mRadius);
        /* 绘制起点的圆形 */
        canvas.drawOval(oval, paint);
      }
      /* mode=2：画路线 */
      else if(mode==2)
      {
        Point point2 = new Point();
        projection.toPixels(gp2, point2);
        paint.setColor(Color.BLACK);
        paint.setStrokeWidth(5);
        paint.setAlpha(120);
        /* 画线 */
        canvas.drawLine(point.x, point.y, point2.x,point2.y, paint);
      }
      /* mode=3：创建终点 */
      else if(mode==3)
      {
        /* 避免误差，先画最后一段的路线 */
        Point point2 = new Point();
        projection.toPixels(gp2, point2);
        paint.setStrokeWidth(5);
        paint.setAlpha(120);
        canvas.drawLine(point.x, point.y, point2.x,point2.y, paint);

        /* 定义 RectF 对象 */
        RectF oval=new RectF(point2.x - mRadius,point2.y - mRadius,
                             point2.x + mRadius,point2.y + mRadius);
        /* 绘制终点的圆形 */
        paint.setAlpha(255);
```

```
        canvas.drawOval(oval, paint);
      }
    }
    return super.draw(canvas, mapView, shadow, when);
  }
}
```

至此，整个演练结束。执行后依次单击“开始记录”和“结束记录”按钮，能实现 GPS 轨迹记录，执行效果如图 17-17 所示。

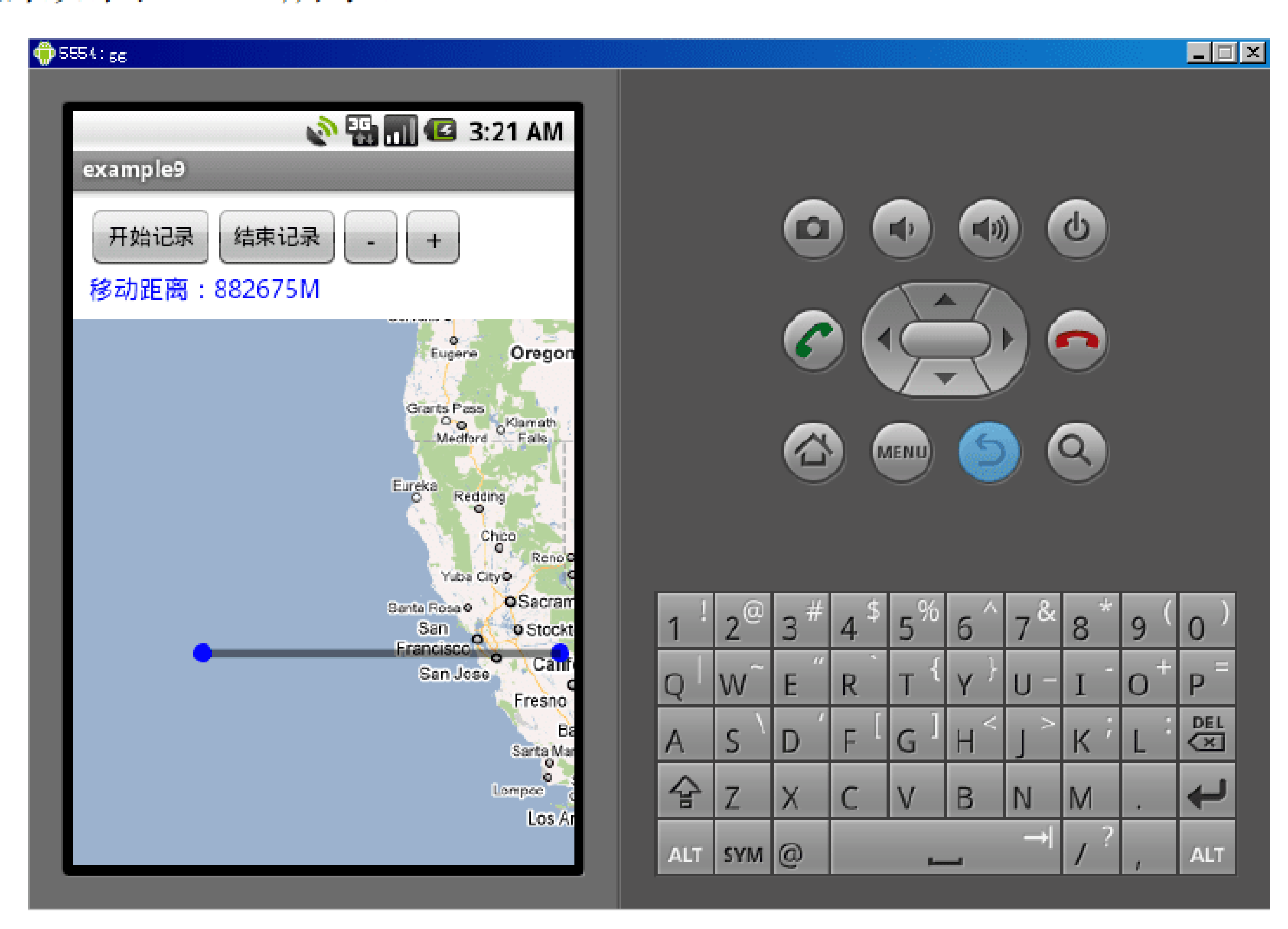


图 17-17　执行效果

17.8　动态二维条码扫描仪

题　目	目　的	源码路径
演练 8	编写动态二维条码扫描程序	“光盘:\daima\17\example11”文件夹

17.8.1　二维码扫描程序

在手机中可以开发一个二维码的扫描程序，这样可以随时随地解码二维条码了。开发二维码的扫描程序需要使用第三方开放的 Library，需要引用 qrcode 项目，在下载.jar 后，将文件名修改为 SourceForgeQRCode.jar，并导入到我们的 Android 工程中去。当前的二维码标准是 QR Code，QR Code 码是由日本 Denso 公司于 1994 年 9 月研制的一种矩阵二维码符号，它具有一维条码及其他二维条码所具有的信息容量大、可靠性高、可表示汉字及图像多种文字信息、保密防伪性强等优点。

17.8.2　具体实现

本实例的主文件是 example11.java，下面是具体的实现流程。

(1) 设置应用程序全屏幕运行，并添加红色正方形红框 View 供 User 对准条形码，然后将创建的红色方框添加到 Activity 中。具体代码如下：

```
@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  /* 使应用程序全屏幕运行，不使用 title bar */
  requestWindowFeature(Window.FEATURE_NO_TITLE);

  setContentView(R.layout.main);
  /* 添加红色正方形红框 View，供 User 对准条形码 */
  DrawCaptureRect mDraw = new DrawCaptureRect
  (
    example2.this,
    110, 10, 100, 100,
    getResources().getColor(R.drawable.lightred)
  );
  /* 将创建的红色方框添加至此 Activity 中 */
  addContentView
  (
    mDraw,
    new LayoutParams
    (
      LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT
    )
  );
```

(2) 分别取得屏幕解析像素，绑定 SurfaceView，设置预览大小。具体代码如下：

```
  /* 取得屏幕解析像素 */
  DisplayMetrics dm = new DisplayMetrics();
  getWindowManager().getDefaultDisplay().getMetrics(dm);
  mImageView01 = (ImageView) findViewById(R.id.myImageView1);
  /* 以 SurfaceView 作为相机 Preview 之用 */
  mSurfaceView01 = (SurfaceView) findViewById(R.id.mSurfaceView1);
  /* 绑定 SurfaceView，取得 SurfaceHolder 对象 */
  mSurfaceHolder01 = mSurfaceView01.getHolder();
  /* Activity 必须实现 SurfaceHolder.Callback */
  mSurfaceHolder01.addCallback(example2.this);
  /* 额外的设置预览大小设置，在此不使用 */
  //mSurfaceHolder01.setFixedSize(320, 240);
  /*
   * 以 SURFACE_TYPE_PUSH_BUFFERS(3)
   * 作为 SurfaceHolder 显示类型
   * */
  mSurfaceHolder01.setType
  (SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

  mButton01 = (Button)findViewById(R.id.myButton1);
  mButton02 = (Button)findViewById(R.id.myButton2);
  mButton03 = (Button)findViewById(R.id.myButton3);
```

(3)　定义方法 mButton01.setOnClickListener，用于打开相机及预览二维条形码。具体代码如下：

```
/* 打开相机及预览二维条形码 */
mButton01.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View arg0)
  {
    // TODO Auto-generated method stub
    /* 自定义初始化打开相机函数 */
    initCamera();
  }
});
```

(4)　定义方法 mButton02.setOnClickListener，用于停止预览。具体代码如下：

```
/* 停止预览 */
mButton02.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View arg0)
  {
    // TODO Auto-generated method stub
    /* 自定义重置相机，并关闭相机预览函数 */
    resetCamera();
  }
});
```

(5)　定义方法 mButton03.setOnClickListener，用于拍照 QR Code 二维条形码。具体代码如下：

```
  /* 拍照 QR Code 二维条形码 */
  mButton03.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View arg0)
    {
      // TODO Auto-generated method stub
      /* 自定义拍照函数 */
      takePicture();
    }
  });
}
```

(6)　定义方法 initCamera()，用于自定义初始相机函数。具体代码如下：

```
/* 自定义初始相机函数 */
private void initCamera()
{
  if(!bIfPreview)
  {
    /* 若相机非在预览模式，则打开相机 */
    mCamera01 = Camera.open();
```

```
    }

    if (mCamera01 != null && !bIfPreview)
    {
      Log.i(TAG, "inside the camera");
      /* 创建 Camera.Parameters 对象 */
      Camera.Parameters parameters = mCamera01.getParameters();
      /* 设置相片格式为 JPEG */
      parameters.setPictureFormat(PixelFormat.JPEG);
      /* 指定 preview 的屏幕大小 */
      parameters.setPreviewSize(160, 120);
      /* 设置图片分辨率大小 */
      parameters.setPictureSize(160, 120);
      /* 将 Camera.Parameters 设置予 Camera */
      mCamera01.setParameters(parameters);
      /* setPreviewDisplay 唯一的参数为 SurfaceHolder */
      mCamera01. setPreviewDisplay (mSurfaceHolder01);
      /* 立即运行 Preview */
      mCamera01.startPreview();
      bIfPreview = true;
    }
  }
```

(7) 定义方法 takePicture()，用于拍照处理并获取图像。具体代码如下：

```
/* 拍照撷取图像 */
private void takePicture()
{
  if (mCamera01 != null && bIfPreview)
  {
    /* 调用 takePicture()方法拍照 */
    mCamera01.takePicture
    (shutterCallback, rawCallback, jpegCallback);
  }
}
```

(8) 定义方法 resetCamera()实现相机重置，在此需要释放 Camera 对象。具体代码如下：

```
/* 相机重置 */
private void resetCamera()
{
  if (mCamera01 != null && bIfPreview)
  {
    mCamera01.stopPreview();
    /*释放 Camera 对象 */
    //mCamera01.release();
    mCamera01 = null;
    bIfPreview = false;
  }
}
private ShutterCallback shutterCallback = new ShutterCallback()
{
  public void onShutter()
```

```
    {
      // Shutter has closed
    }
  };
  private PictureCallback rawCallback = new PictureCallback()
  {
    public void onPictureTaken(byte[] _data, Camera _camera)
    {
      // TODO Handle RAW image data
    }
  };
```

(9)　定义方法 onPictureTaken 对传入的图片进行处理。具体流程如下。

第 1 步：设置 onPictureTaken 传入的第一个参数即为相片的 byte；

第 2 步：使用 Matrix.postScale 方法缩小 Bitmap Size；

第 3 步：创建新的 Bitmap 对象，并获取 4∶3 图片的居中红色框部分 100×100 像素；

第 4 步：将拍照的图文件以 ImageView 显示出来，并将传入的图文件译码成字符串；

第 5 步：定义方法 mMakeTextToast 输出提示。

具体代码如下：

```
  private PictureCallback jpegCallback = new PictureCallback()
  {
    public void onPictureTaken(byte[]  data, Camera  camera)
    {
      // TODO Handle JPEG image data
      try
      {
        /* onPictureTaken 传入的第一个参数即为相片的 byte */
        Bitmap bm =
        BitmapFactory.decodeByteArray(_data, 0, _data.length);
        int resizeWidth = 160;
        int resizeHeight = 120;
        float scaleWidth = ((float) resizeWidth) / bm.getWidth();
        float scaleHeight = ((float) resizeHeight) / bm.getHeight();
        Matrix matrix = new Matrix();
        /* 使用 Matrix.postScale 方法缩小 Bitmap Size*/
        matrix.postScale(scaleWidth, scaleHeight);
        /* 创建新的 Bitmap 对象 */
        Bitmap resizedBitmap = Bitmap.createBitmap
        (bm, 0, 0, bm.getWidth(), bm.getHeight(), matrix, true);
        /* 撷取 4:3 图片的居中红色框部分 100×100 像素 */
        Bitmap resizedBitmapSquare = Bitmap.createBitmap
        (resizedBitmap, 30, 10, 100, 100);
        /* 将拍照的图文件以 ImageView 显示出来 */
        mImageView01.setImageBitmap(resizedBitmapSquare);
        /* 将传入的图文件译码成字符串 */
        String strQR2 = decodeQRImage(resizedBitmapSquare);
        if(strQR2!="")
        {
          if (URLUtil.isNetworkUrl(strQR2))
          {
```

```
          /* OMIA 规范，网址条形码，打开浏览器上网 */
          mMakeTextToast(strQR2, true);
          Uri mUri = Uri.parse(strQR2);
          Intent intent = new Intent(Intent.ACTION_VIEW, mUri);
          startActivity(intent);
        }
        else if(eregi("wtai://",strQR2))
        {
          /* OMIA 规范，手机拨打电话格式 */
          String[] aryTemp01 = strQR2.split("wtai://");
          Intent myIntentDial = new Intent
          (
            "android.intent.action.CALL",
            Uri.parse("tel:"+aryTemp01[1])
          );
          startActivity(myIntentDial);
        }
        else if(eregi("TEL:",strQR2))
        {
          /* OMIA 规范，手机拨打电话格式 */
          String[] aryTemp01 = strQR2.split("TEL:");
          Intent myIntentDial = new Intent
          (
            "android.intent.action.CALL",
            Uri.parse("tel:"+aryTemp01[1])
          );
          startActivity(myIntentDial);
        }
        else
        {
          /* 若仅是文字，则以 Toast 显示出来 */
          mMakeTextToast(strQR2, true);
        }
      }
      /* 显示完图文件，立即重置相机，并关闭预览 */
      resetCamera();

      /* 再重新启动相机继续预览 */
      initCamera();
    }
    catch (Exception e)
    {
      Log.e(TAG, e.getMessage());
    }
  }
};
public void mMakeTextToast(String str, boolean isLong)
{
  if(isLong==true)
  {
    Toast.makeText(example2.this, str, Toast.LENGTH_LONG).show();
  }
```

```
  else
  {
    Toast.makeText(example2.this, str, Toast.LENGTH_SHORT).show();
  }
}
```

(10) 定义方法 checkSDCard()，用于判断记忆卡是否存在。具体代码如下：

```
private boolean checkSDCard()
{
  /* 判断记忆卡是否存在 */
  if(android.os.Environment.getExternalStorageState().equals
  (android.os.Environment.MEDIA_MOUNTED))
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

(11) 定义方法 decodeQRImage(Bitmap myBmp)，用于解码传入的 Bitmap 图片。具体代码如下：

```
/* 解码传入的 Bitmap 图片 */
public String decodeQRImage(Bitmap myBmp)
{
  String strDecodedData = "";
  try
  {
    QRCodeDecoder decoder = new QRCodeDecoder();
    strDecodedData  = new String
    (decoder.decode(new AndroidQRCodeImage(myBmp)));
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
  return strDecodedData;
}
```

(12) 自定义实现 QRCodeImage 类 AndroidQRCodeImage。具体代码如下：

```
/* 自定义实现 QRCodeImage 类 */
class AndroidQRCodeImage implements QRCodeImage
{
  Bitmap image;
  public AndroidQRCodeImage(Bitmap image)
  {
    this.image = image;
  }
  public int getWidth()
  {
```

```
      return image.getWidth();
    }
    public int getHeight()
    {
      return image.getHeight();
    }

    public int getPixel(int x, int y)
    {
      return image.getPixel(x, y);
    }
  }
```

(13) 定义类 DrawCaptureRect，用于绘制相机预览画面里的正方形方框。具体代码如下：

```
  /* 绘制相机预览画面里的正方形方框 */
  class DrawCaptureRect extends View
  {
    private int colorFill;
    private int intLeft,intTop,intWidth,intHeight;

    public DrawCaptureRect
    (
      Context context, int intX, int intY, int intWidth,
      int intHeight, int colorFill
    )
    {
      super(context);
      this.colorFill = colorFill;
      this.intLeft = intX;
      this.intTop = intY;
      this.intWidth = intWidth;
      this.intHeight = intHeight;
    }
    @Override
    protected void onDraw(Canvas canvas)
    {
      Paint mPaint01 = new Paint();
      mPaint01.setStyle(Paint.Style.FILL);
      mPaint01.setColor(colorFill);
      mPaint01.setStrokeWidth(1.0F);
      /* 在画布上绘制红色的四条方边框作为瞄准器 */
      canvas.drawLine
      (
        this.intLeft, this.intTop,
        this.intLeft+intWidth, this.intTop, mPaint01
      );
      canvas.drawLine
      (
        this.intLeft, this.intTop,
        this.intLeft, this.intTop+intHeight, mPaint01
      );
```

```
    canvas.drawLine
    (
      this.intLeft+intWidth, this.intTop,
      this.intLeft+intWidth, this.intTop+intHeight, mPaint01
    );
    canvas.drawLine
    (
      this.intLeft, this.intTop+intHeight,
      this.intLeft+intWidth, this.intTop+intHeight, mPaint01
    );
    super.onDraw(canvas);
  }
}
```

(14) 定义方法 eregi(String strPat, String strUnknow)，用于实现自定义比较字符串处理。具体代码如下：

```
 /* 自定义比较字符串函数 */
 public static boolean eregi(String strPat, String strUnknow)
 {
   String strPattern = "(?i)"+strPat;
   Pattern p = Pattern.compile(strPattern);
   Matcher m = p.matcher(strUnknow);
   return m.find();
 }
 @Override
 public void surfaceChanged
 (SurfaceHolder surfaceholder, int format, int w, int h)
 {
   // TODO Auto-generated method stub
   Log.i(TAG, "Surface Changed");
 }
 @Override
 public void surfaceCreated(SurfaceHolder surfaceholder)
 {
   // TODO Auto-generated method stub
   Log.i(TAG, "Surface Changed");
 }
 @Override
 public void surfaceDestroyed(SurfaceHolder surfaceholder)
 {
   // TODO Auto-generated method stub
   Log.i(TAG, "Surface Destroyed");
 }
 @Override
 protected void onPause()
 {
   // TODO Auto-generated method stub
   super.onPause();
 }
}
```

至此，整个演练结束，执行后能够通过手机拍照实现二维码解析，执行结果如图 17-18 所示。

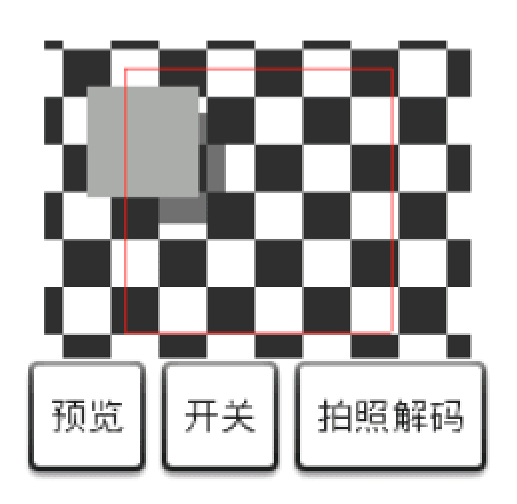


图 17-18　执行效果

17.9　设置手机屏幕颜色

题　目	目　的	源码路径
演练 9	用 Color 类和 Paint 类实现绘图处理	“光盘:\daima\17\example12”文件夹

17.9.1　屏幕的显示颜色

在现实应用中，可以设计屏幕的显示颜色，在 Android 中，可以通过 Android.os.PowerManage 控制手机的 WakeLock，这样可以让手机的屏幕保持在恒亮状态，再通过程序将手机亮度调到最高 255。

17.9.2　具体实现

本实例的主文件是 example3.java 和 MyAdapter，下面是具体的实现流程。

1. 文件 example3.java

在文件 example3.java 中，先将屏幕设置为全屏显示，然后以 PowerManager.newWakeLock 来获取 WakeLock 对象，并记下 Activity 启动前的屏幕亮度。当启动 Activity 时调用 onResume()，并运行 wakeLock 方法，设置屏幕亮度为 255；当暂停或停止 Activity 时调用 onPause()，并运行 wakeUnlock 方法，设置屏幕亮度为程序启动时的亮度。

文件 example3.java 的具体代码如下：

```
public class example3 extends Activity
{
  private boolean ifLocked = false;
  private PowerManager.WakeLock mWakeLock;
  private PowerManager mPowerManager;
  private LinearLayout mLinearLayout;
  static final private int M_CHOOSE = Menu.FIRST;
  static final private int M_EXIT = Menu.FIRST+1;
  private int[] color={R.drawable.white,R.drawable.blue,
                   R.drawable.pink,R.drawable.green,
                   R.drawable.orange,R.drawable.yellow};
  private int[] text={R.string.str white,R.string.str blue,
                  R.string.str_pink,R.string.str_green,
```

```
                    R.string.str orange,R.string.str yellow};

@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  requestWindowFeature(Window.FEATURE_NO_TITLE);

  this.getWindow().setFlags
  (
    WindowManager.LayoutParams.FLAG_FULLSCREEN,
    WindowManager.LayoutParams.FLAG FULLSCREEN
  );
  setContentView(R.layout.main);

  /* Activity 启动时将屏幕调整为最亮 */
  WindowManager.LayoutParams lp = getWindow().getAttributes();
  lp.screenBrightness = 1.0f;
  getWindow().setAttributes(lp);

  /* 初始化 mLinearLayout */
  mLinearLayout=(LinearLayout)findViewById(R.id.myLinearLayout1);

  /* 取得 PowerManager */
  mPowerManager = (PowerManager)
              getSystemService(Context.POWER SERVICE);
  /* 取得 WakeLock */
  mWakeLock = mPowerManager.newWakeLock
  (
    PowerManager.SCREEN_BRIGHT_WAKE_LOCK, "BackLight"
  );
}

@Override
public boolean onCreateOptionsMenu(Menu menu)
{
  /* menu 群组 ID */
  int idGroup1 = 0;
  /* menuItemID */
  int orderMenuItem1 = Menu.NONE;
  int orderMenuItem2 = Menu.NONE+1;
  /* 建立 menu */
  menu.add(idGroup1,M_CHOOSE,orderMenuItem1,R.string.str_title);
  menu.add(idGroup1,M_EXIT,orderMenuItem2,R.string.str_exit);
  menu.setGroupCheckable(idGroup1, true, true);

  return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item)
```

```
{
  switch(item.getItemId())
  {
    case (M_CHOOSE):
      /*选择背景颜色的 AlertDialog */
      new AlertDialog.Builder(example3.this)
        .setTitle(getResources().getString(R.string.str_title))
        .setAdapter(new MyAdapter(this,color,text),listener1)
        .setPositiveButton("取消",
            new DialogInterface.OnClickListener()
        {
          public void onClick(DialogInterface dialog, int which)
          {
          }
        })
        .show();
      break;
    case (M_EXIT):
      /* 离开程序 */
      this.finish();
      break;
  }
  return super.onOptionsItemSelected(item);
}
/* 选择背景颜色的 AlertDialog 的 OnClickListener */
OnClickListener listener1=new DialogInterface.OnClickListener()
{
  public void onClick(DialogInterface dialog,int which)
  {
    /* 更改背景颜色 */
    mLinearLayout.setBackgroundResource(color[which]);
    /* Toast 显示设定的颜色 */
    Toast.makeText(example3.this,
                getResources().getString(text[which]),
                Toast.LENGTH_LONG).show();
  }
};

@Override
protected void onResume()
{
  /* onResume()时呼叫 wakeLock() */
  wakeLock();
  super.onResume();
}

@Override
protected void onPause()
{
  /* onPause()时呼叫 wakeUnlock() */
  wakeUnlock();
  super.onPause();
```

```
  }
  /* 唤起 WakeLock 的 method */
  private void wakeLock()
  {
    if (!ifLocked)
    {
      ifLocked = true;
      mWakeLock.acquire();
    }
  }
  /* 释放 WakeLock 的 method */
  private void wakeUnlock()
  {
    if (ifLocked)
    {
      mWakeLock.release();
      ifLocked = false;
    }
  }
}
```

2. 文件 MyAdapter.java

在文件 MyAdapter.java 中，设置了背景颜色菜单 Adapter 继承来自 android.widget.BaseAdapter，使用 change_color.xml 作为 Layout。具体代码如下：

```
/* 定义 Adapter，继承 android.widget.BaseAdapter */
public class MyAdapter extends BaseAdapter
{
  private LayoutInflater mInflater;
  private int[] color;
  private int[] text;
  public MyAdapter(Context context,int[]  color,int[]  text)
  {
    mInflater = LayoutInflater.from(context);
    color = _color;
    text = _text;
  }
  @Override
  public int getCount()
  {
    return text.length;
  }
  @Override
  public Object getItem(int position)
  {
    return text[position];
  }
  @Override
  public long getItemId(int position)
  {
    return position;
```

```
  }
  @Override
  public View getView(int position,View convertView,ViewGroup par)
  {
    ViewHolder holder;
    if(convertView == null)
    {
      convertView = mInflater.inflate(R.layout.change_color, null);
      /* 初始化 holder 的 text */
      holder = new ViewHolder();
      holder.mText=(TextView)convertView.findViewById(R.id.myText);
      convertView.setTag(holder);
    }
    else
    {
      holder = (ViewHolder) convertView.getTag();
    }
    holder.mText.setText(text[position]);
    holder.mText.setBackgroundResource(color[position]);

    return convertView;
  }
  /* class ViewHolder */
  private class ViewHolder
  {
    TextView mText;
  }
}
```

至此，整个演练结束，执行后将按照默认样式显示屏幕颜色，默认效果如图 17-19 所示；单击 MENU 按钮后弹出两个标签，如图 17-20 所示；单击“选择背光颜色”标签后弹出设置对话框，在此可以设置要显示的颜色，如图 17-21 所示。

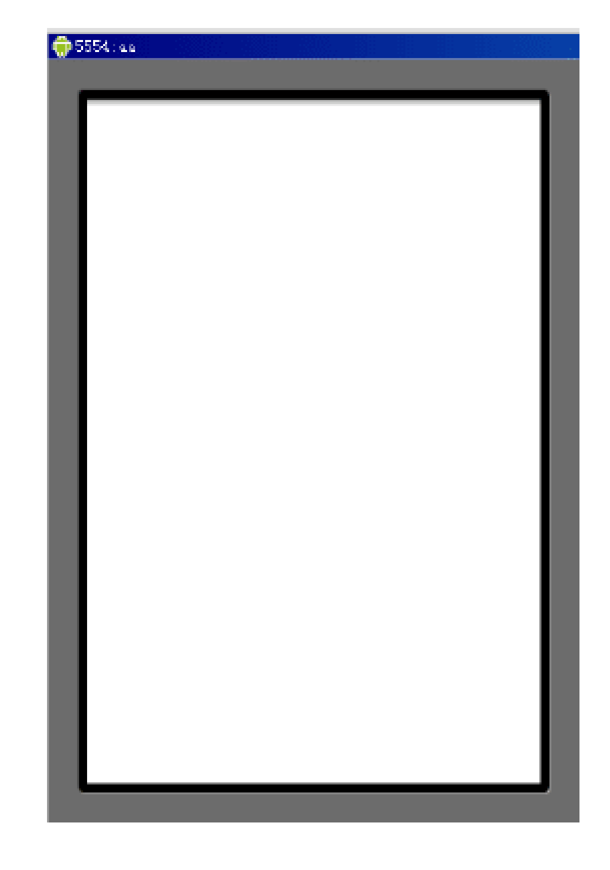


图 17-19　默认效果

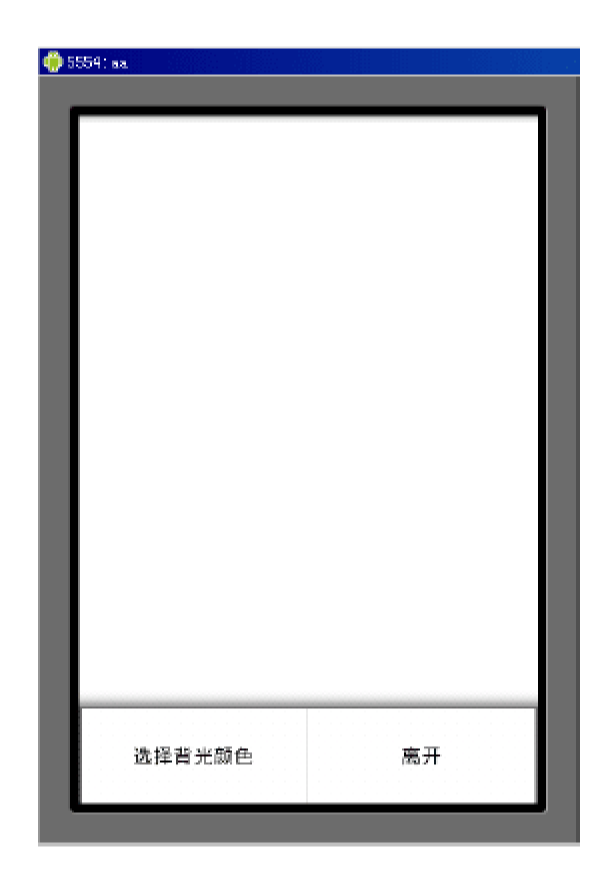


图 17-20　弹出的两个标签



图 17-21　设置对话框



第 18 章　开发 RSS 阅读器

RSS 是一个开放的新闻模式，RSS(也叫聚合内容，Really Simple Syndication)是在线共享内容的一种简易方式。通常在时效性比较强的内容上使用 RSS 订阅能更快速获取信息，网站提供 RSS 输出，有利于让用户获取网站内容的最新更新。在本章的内容中，将通过一个具体实例的实现过程，介绍在 Android 中查看指定 RSS 新闻的基本流程。

18.1　RSS 风云再起

无论是在当今社会还是在古代社会，信息都十分重要。古代因为信息传递技术的限制，所以很难及时获取某些信息。随着科学技术的发展，最近几年兴起的 RSS 技术已经深入人心。在本节的内容中，将简要讲解 RSS 用途、RSS 阅读器和 RSS 的基本语法知识，为 RSS 开发打好基础。

18.1.1　RSS 的用途

在讲解 RSS 系统的实现流程前，先来了解一下 RSS 的重要用途。RSS 的主要用途如下。

(1)　可以订阅 Blog(你可以订阅工作中所需要的技术文章；也可以订阅与你有共同爱好的作者的 Blog，总之，你对什么感兴趣就可以订阅什么)。

(2)　订阅新闻(无论是奇闻怪事、明星消息还是体坛风云，只要你想知道的，都可以订阅)。

(3)　不需要一个网站一个网站，一个网页一个网页去逛了。只要将你需要的内容订阅在一个 RSS 阅读器中，这些内容就会自动出现在你的阅读器里，你也不必为了一个急切想知道的消息而不断地刷新网页，因为一旦有了更新，RSS 阅读器就会自动通知你。

订阅 RSS 新闻需要先安装一个 RSS 阅读器，然后将提供 RSS 服务的网站加入到 RSS 阅读器的频道即可，具体流程如下：

(1)　选择有价值的 RSS 信息源；

(2)　启动 RSS 订阅程序，将信息源添加到自己的 RSS 阅读器或者在线 RSS；

(3)　接收并获取定制的 RSS 信息。

18.1.2 RSS 阅读器

目前，RSS 阅读器基本可以分为以下三类。

(1) 大多数阅读器是运行在计算机桌面上的应用程序，通过所订阅网站的新闻供应，可自动、定时地更新新闻标题。在该类阅读器中，有 Awasu、FeedDemon 和 RSSReader 这三款流行的阅读器，都提供免费试用版和付费高级版。国内最近也推出了几款 RSS 阅读器：周博通、看天下、博阅。另外，开源社区也推出了很多优秀的阅读器，例如 RSSOWl(完全 Java 开发)它不仅完全支持中文界面，而且还是完全的免费软件。

(2) 新闻阅读器通常是内嵌于已在计算机中运行的应用程序中。例如，NewsGator 内嵌在微软的 Outlook 中，所订阅的新闻标题位于 Outlook 的收件箱文件夹中。

(3) 在线 Web RSS 阅读器，其优势在于不需要安装任何软件就可以获得 RSS 阅读的便利，并且可以保存阅读状态，推荐和收藏自己感兴趣的文章。提供此服务的有两类网站，一类是专门提供 RSS 阅读器的网站，例如，鲜果、抓虾；另一种是提供个性化首页的网站，例如，国外的 netvibes、pageflakes；国内的雅蛙、阔地。

18.1.3 RSS 语法

RSS 2.0 的语法规则非常简单且十分严格，例如下面的代码：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">
<channel>
<title>W3Schools</title>
<link>http://www.w3schools.com</link>
<description>W3Schools Web Tutorials </description>
<item>
<title>RSS Tutorial</title>
<link>http://www.w3schools.com/rss</link>
<description>Check out the RSS tutorial
on W3Schools.com</description>
</item>
</channel>
</rss>
```

其中，<channel>元素内是描述 RSS feed 的地方。

RSS 的<channel>元素是项目内容显示的地方。它就像 RSS 的标题，一般来讲它不会频繁地改动。有三个内部元素是必须有的，分别是<title>、<link>和<description>。具体说明如下。

- <title>元素里应该包含你的站和你的 RSS feed 简短说明。
- <link>元素应该定义你的网站主页的链接。
- <description>元素应该描述你的 RSS feed。

<channel>内的可选元素如下。

- <category>：定义一个或多个频道分类。
- <cloud>：允许更新通告。

- <copyright>：提醒有关版权。
- <docs>：频道所使用的 RSS 版本文档 URL。
- <generator>：如果频道是自动生成器产生的，就在这里定义。
- <image>：给频道加图片。
- <language>：描述频道所使用的语言。
- <lastBuildDate>：定义频道最近一次改动的时间。
- <managingEditor>：定义编辑站点人员的 E-mail 地址。
- <pubDate>：定义频道最新的发布时间。
- <rating>：页面评估。
- <ttl>：存活的有效时间。
- <webMaster>定义站长的邮件地址。

<item>元素内是你网站链接和描述更新内容的地方，<item>是显示 RSS 更新内容的地方，它像文章的标题。当你的站点有更新时 RSSfeed 中的<item>元素就会被建立起来，<item>元素里有几个可选的元素，但<title>和<description>是必须要有的。

一个 RSS 的<item>应该包括：<title>、<link>和<description>三个元素，具体说明如下。

- <title>元素是项目的题目，应该用十分简短的语言描述。
- <link>元素是项目所关联的链接。
- <description>元素就是 RSS feed 的描述部分，描述你的 RSS feed 项目。

<item>可选的元素如下。

- <author>：定义作者。
- <category>：类别。
- <comments>：针对项目评论页的 URL。
- <enclosure>：描述一个与项目有关的媒体对象。
- <guid>：针对项目定义独特的标志。
- <pubDate>：项目的发布时间。
- <source>：转载地址(源地址)。

在<description>中建议使用<![CDATA[ ]]>，所有在 CDATA 部件之间的文本都会被解析器忽略。

> **注意：** CDATA 部件之间不能再包含 CDATA 部件(不能嵌套)。如果 CDATA 部件包含了字符"]]>"或者 CDATA，将很有可能出错。同样，也要注意在字符串"]]>"之间没有空格或者换行符。

18.2 实现流程

本项目实例的功能是在手机中显示指定的 RSS 信息，项目的具体实现流程如图 18-1 所示。

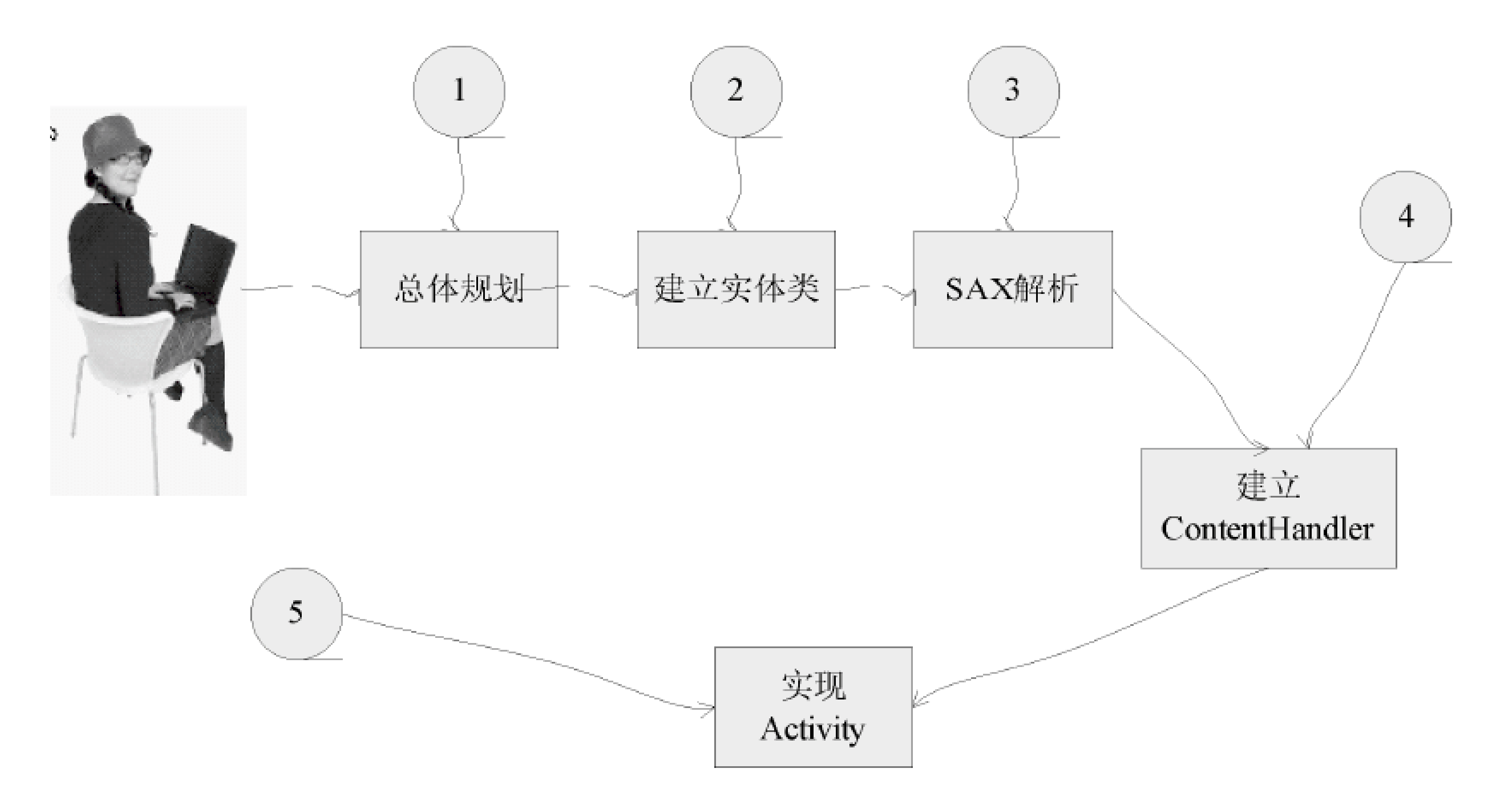


图 18-1　实现流程图

18.3　具 体 实 现

在使用 RSS 订阅时，通常通过网站提供的“订阅 RSS”链接或小图标实现，当单击链接后，会弹出包含 RSS 内容的页面，此页面的网址是网站的 RSS 网址。当连接到这个网址后，服务器端会返回 RSS 标准规格的 XML 文件，只要按照统一格式来解析 XML 文件，就可以得到 RSS 内的相关信息。

在本实例中，用户只需要输入一个 RSS Feed 网址，通过 SAXParser 解析后就可以直接在手机上浏览在线实时新闻。本实例的主程序文件是 example10.java、example10_1.java、example10_2.java、News.java、MyHandler.java 和 MyAdapter.java，下面分别介绍其实现流程。

18.3.1　主程序 example10.java

主程序 example10.java 中以 EditText 来作为输入 RSS 连接组件，当输入网址后，单击“解析”按钮后，按钮的 onClick 事件会被触发，运行 EditText 的空白检查。当检查无误后，将输入的网址写入 Bundle 对象中，再将 Bundle 对象 assign 给 Intent，并通过 startActivityForResult()来触发 example10_1 的 Activity。

主程序 example10.java 的主要实现代码如下：

```
public class example10 extends Activity
{
  /* 变量声明 */
  private Button mButton;
  private EditText mEditText;
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /* 初始化对象 */
    mEditText=(EditText) findViewById(R.id.myEdit);
```

```
  mButton=(Button) findViewById(R.id.myButton);
  /* 设置 Button 的 onClick 事件 */
  mButton.setOnClickListener(new Button.OnClickListener()
  {
    @Override
    public void onClick(View v)
    {
      String path=mEditText.getText().toString();
      if(path.equals(""))
      {
        showDialog("网址不可为空白!");
      }
      else
      {
        /* new 一个 Intent 对象，并指定 class */
        Intent intent = new Intent();
        intent.setClass(example10.this,example10_1.class);

        /* new 一个 Bundle 对象，并将要传递的数据传入 */
        Bundle bundle = new Bundle();
        bundle.putString("path",path);
        /* 将 Bundle 对象 assign 给 Intent */
        intent.putExtras(bundle);
        /* 调用 Activity EX08_13_1 */
        startActivityForResult(intent,0);
      }
    }
  });
}
/* 覆盖 onActivityResult()*/
@Override
protected void onActivityResult(int requestCode,int resultCode, Intent data)
{
  switch (resultCode)
  {
    case 99:
      /* 返回错误时以 Dialog 显示 */
      Bundle bunde = data.getExtras();
      String error = bunde.getString("error");
      showDialog(error);
      break;
    default:
      break;
   }
 }

/* 显示 Dialog 的方法 */
private void showDialog(String mess){
  new AlertDialog.Builder(example10.this).setTitle("Message")
  .setMessage(mess)
  .setNegativeButton("确定", new DialogInterface.OnClickListener()
  {
```

```
      public void onClick(DialogInterface dialog, int which)
      {
      }
    })
    .show();
  }
}
```

18.3.2 文件 example10_1.java

文件 example10_1.java 是一个 ListActivity，是通过主程序 example10.java 来调用的，用于显示订阅的 RSS 内容列表，其实现流程如下。

(1) 设置 layout 为 newslist.xml，取得 Intent 中的 Bundle 对象，并取得 Bundle 对象中的数据，然后调用 getRss()取得解析后的 List。具体代码如下：

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  /* 设置 layout 为 newslist.xml */
  setContentView(R.layout.newslist);
  mText=(TextView) findViewById(R.id.myText);
  /* 取得 Intent 中的 Bundle 对象 */
  Intent intent=this.getIntent();
  Bundle bunde = intent.getExtras();
  /* 取得 Bundle 对象中的数据 */
  String path = bunde.getString("path");
  /* 调用 getRss()取得解析后的 List */
  li=getRss(path);
  mText.setText(title);
  /* 设置自定义的 MyAdapter */
  setListAdapter(new MyAdapter(this,li));
}
```

(2) 定义 onListItemClick，定义监听 ListItem 被单击时要做的动作，具体流程如下。

第 1 步：获取 News 对象，新建一个 Intent 对象并指定其 class。

第 2 步：新建一个 Bundle 对象，将要传递的数据传入。

第 3 步：将 Bundle 对象 assign 给 Intent，并调用 Activity example10_2。

具体代码如下：

```
/* 设置 ListItem 被点击时要做的动作 */
@Override
protected void onListItemClick(ListView l,View v,int position,long id)
{
  /* 取得 News 对象 */
  News ns=(News)li.get(position);
  /* new 一个 Intent 对象，并指定 class */
  Intent intent = new Intent();
  intent.setClass(example10 1.this,example10 2.class);
  /* new 一个 Bundle 对象，并将要传递的数据传入 */
  Bundle bundle = new Bundle();
```

```
  bundle.putString("title",ns.getTitle());
  bundle.putString("desc",ns.getDesc());
  bundle.putString("link",ns.getLink());
  /* 将 Bundle 对象 assign 给 Intent */
  intent.putExtras(bundle);
  /* 调用 Activity example10_2 */
  startActivity(intent);
}
```

(3)　定义 getRss(String path)，用于解析 XML，具体流程如下。

第 1 步：通过 URL 获取地址，并创建 SAXParser 对象和 XMLReader 对象。

第 2 步：设置自定义的 MyHandler 给 XMLReader，并解析 XML。

第 3 步：取得 RSS 标题与内容列表。

具体代码如下：

```
/* 解析 XML 的方法 */
private List<News> getRss(String path)
{
  List<News> data=new ArrayList<News>();
  URL url = null;
  try
  {
    url = new URL(path);
    /* 产生 SAXParser 对象 */
    SAXParserFactory spf = SAXParserFactory.newInstance();
    SAXParser sp = spf.newSAXParser();
    /* 产生 XMLReader 对象 */
    XMLReader xr = sp.getXMLReader();
    /* 设置自定义的 MyHandler 给 XMLReader */
    MyHandler myExampleHandler = new MyHandler();
    xr.setContentHandler(myExampleHandler);
    /* 解析 XML */
    xr.parse(new InputSource(url.openStream()));
    /* 取得 RSS 标题与内容列表 */
    data =myExampleHandler.getParsedData();
    title=myExampleHandler.getRssTitle();
  }
```

(4)　有异常时则输出错误提示对话框。具体代码如下：

```
  catch (Exception e)
  {
    /* 发生错误时返回 result */
    Intent intent=new Intent();
    Bundle bundle = new Bundle();
    bundle.putString("error",""+e);
    intent.putExtras(bundle);
    /* 错误的返回值设置为 99 */
    example10_1.this.setResult(99, intent);
    example10_1.this.finish();
  }
  return data;
```

```
  }
}
```

18.3.3　文件 example10_2.java

文件 example10_2.java 由 example10_1 唤起，用于显示上一个 Activity 所单击的新闻内容。当程序被唤起后，会首先从 Bundle 对象中获取 News 的 title、link 和 desc 并显示在画面中。以 Linkify.addLinks()将 link 设置为一个 WEB_URLS 形式的链接。当用户单击链接后，会通过设置的网址直接打开 Web 浏览器来浏览网页。其主要实现代码如下：

```
public class example10_2 extends Activity
{
  /* 变量声明 */
  private TextView mTitle;
  private TextView mDesc;
  private TextView mLink;
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    /* 设置 layout 为 newscontent.xml */
    setContentView(R.layout.newscontent);
    /* 初始化对象 */
    mTitle=(TextView) findViewById(R.id.myTitle);
    mDesc=(TextView) findViewById(R.id.myDesc);
    mLink=(TextView) findViewById(R.id.myLink);
    /* 取得 Intent 中的 Bundle 对象 */
    Intent intent=this.getIntent();
    Bundle bunde = intent.getExtras();
    /* 取得 Bundle 对象中的数据 */
    mTitle.setText(bunde.getString("title"));
    mDesc.setText(bunde.getString("desc")+"....");
    mLink.setText(bunde.getString("link"));
    /* 设置 mLink 为网页链接 */
    Linkify.addLinks(mLink,Linkify.WEB_URLS);
  }
}
```

18.3.4　文件 News.java

文件 News.java 是一个 JavaBean 类，用于存放每一篇新闻信息。每一个 News 对象代表了一条新闻，在 News 对象中定义了新闻的标题、描述、网站链接和发布时间这四个属性。JavaBean 类中的方法都是以 setAAA()和 getAAA()方式来命名的，所以用 setAAA()来设置属性值或通过 getAAA()来获取属性值。具体代码如下：

```
package irdc.example10;

public class News
{
  /* 新建每条新闻的 4 条属性对象 */
```

```
  private String  title="";
  private String _link="";
  private String _desc="";
  private String _date="";
  public String getTitle()
  {
    return _title;
  }
  public String getLink()
  {
    return _link;
  }
  public String getDesc()
  {
    return _desc;
  }
  public String getDate()
  {
    return _date;
  }
  public void setTitle(String title)
  {
    _title=title;
  }
  public void setLink(String link)
  {
    _link=link;
  }
  public void setDesc(String desc)
  {
    _desc=desc;
  }
  public void setDate(String date)
  {
    _date=date;
  }
}
```

18.3.5　文件 MyAdapter.java

在文件 MyAdapter.java 中定义了 Adapter 对象，它继承自 android.widget.BaseAdapter，用于设置 ListView 中要显示的信息，以 news_row.xml 作为 Layout。主要代码如下：

```
/* 自定义的 Adapter，继承 android.widget.BaseAdapter */
public class MyAdapter extends BaseAdapter
{
  /* 变量声明  */
  private LayoutInflater mInflater;
  private List<News> items;
  /* MyAdapter 的构造器，传递两个参数  */
  public MyAdapter(Context context,List<News> it)
```

```
{
  /* 参数初始化 */
  mInflater = LayoutInflater.from(context);
  items = it;
}
/* 因继承 BaseAdapter，需重写以下方法 */
@Override
public int getCount()
{
  return items.size();
}
@Override
public Object getItem(int position)
{
  return items.get(position);
}
@Override
public long getItemId(int position)
{
  return position;
}
@Override
public View getView(int position,View convertView,ViewGroup par)
{
  ViewHolder holder;

  if(convertView == null)
  {
    /* 使用自定义的 news_row 作为 Layout */
    convertView = mInflater.inflate(R.layout.news_row, null);
    /* 初始化 holder 的 text 与 icon */
    holder = new ViewHolder();
    holder.text = (TextView) convertView.findViewById(R.id.text);
    convertView.setTag(holder);
  }
  else
  {
    holder = (ViewHolder) convertView.getTag();
  }
  News tmpN=(News)items.get(position);
  holder.text.setText(tmpN.getTitle());

  return convertView;
}

/* class ViewHolder */
private class ViewHolder
{
  TextView text;
}
}
```

18.3.6　文件 MyHandler.java

在文件 MyHandler.java 中定义了 MyHandler 对象，它继承自 org.xml.sax.helpers.DefaultHandler，用于解析 XML 文件并获取对应的信息，下面是文件 MyHandler.java 的具体实现流程。

(1)　分别将转换成 List<News>的 XML 数据返回，将解析出的 RSS title 返回。然后调用 startDocument()开始解析操作。当解析结束时，调用 endDocument()方法，当解析到 Element 开头时，调用 startElement()方法。具体代码如下：

```
/* 将转换成 List<News>的 XML 数据返回 */
public List<News> getParsedData()
{
  return li;
}
/* 将解析出的 RSS title 返回 */
public String getRssTitle()
{
  return title;
}
/* XML 文件开始解析时调用此方法 */
@Override
public void startDocument() throws SAXException
{
  li = new ArrayList<News>();
}
/* XML 文件结束解析时调用此方法 */
@Override
public void endDocument() throws SAXException
{
}
/* 解析到 Element 的开头时调用此方法 */
@Override
public void startElement(String namespaceURI, String localName,
           String qName, Attributes atts) throws SAXException
{
  if (localName.equals("item"))
  {
    this.in_item = true;
    /* 解析到 item 的开头时 new 一个 News 对象 */
    news=new News();
  }
  else if (localName.equals("title"))
  {
    if(this.in_item)
    {
      this.in_title = true;
    }
    else
    {
      this.in_mainTitle = true;
    }
  }
```

```
    else if (localName.equals("link"))
    {
      if(this.in_item)
      {
        this.in_link = true;
      }
    }
    else if (localName.equals("description"))
    {
      if(this.in_item)
      {
        this.in desc = true;
      }
    }
    else if (localName.equals("pubDate"))
    {
      if(this.in_item)
      {
        this.in_date = true;
      }
    }
  }
```

(2) 当解析到 Element 的结尾时调用 endElement()方法，其实现流程如下。

第 1 步：解析到 item 的结尾时将 News 对象写入 List 中。

第 2 步：根据 Item 选项分别设置 News 对象的 title、设置 RSS 的 title、设置 News 对象的 link、设置 News 对象的 description 和 pubDate。

具体代码如下：

```
/* 解析到 Element 的结尾时调用此方法 */
@Override
public void endElement(String namespaceURI, String localName,
                       String qName) throws SAXException
{
  if (localName.equals("item"))
  {
    this.in_item = false;
    /* 解析到 item 的结尾时将 News 对象写入 List 中 */
    li.add(news);
  }
  else if (localName.equals("title"))
  {
    if(this.in item)
    {
      /* 设置 News 对象的 title */
      news.setTitle(buf.toString().trim());
      buf.setLength(0);
      this.in_title = false;
    }
    else
    {
```

```
        /* 设置 RSS 的 title */
        title=buf.toString().trim();
        buf.setLength(0);
        this.in_mainTitle = false;
      }
    }
    else if (localName.equals("link"))
    {
      if(this.in_item)
      {
        /* 设置 News 对象的 link */
        news.setLink(buf.toString().trim());
        buf.setLength(0);
        this.in_link = false;
      }
    }
    else if (localName.equals("description"))
    {
      if(in_item)
      {
        /* 设置 News 对象的 description */
        news.setDesc(buf.toString().trim());
        buf.setLength(0);
        this.in_desc = false;
      }
    }
    else if (localName.equals("pubDate"))
    {
      if(in_item)
      {
        /* 设置 News 对象的 pubDate */
        news.setDate(buf.toString().trim());
        buf.setLength(0);
        this.in_date = false;
      }
    }
  }
```

(3) 定义方法 characters()，用于获取 Element 开头和结尾中间的字符串。具体代码如下：

```
  /* 取得 Element 的开头结尾中间夹的字符串 */
  @Override
  public void characters(char ch[], int start, int length)
  {
    if(this.in_item||this.in_mainTitle)
    {
      /* 将 char[]添加 StringBuffer */
      buf.append(ch,start,length);
    }
  }
}
```

至此，整个实例讲解完毕，执行后的效果如图 18-2 所示。在文本框中输入 RSS 网址

http://rss.sina.com.cn/news/marquee/ddt.xml，然后单击“开始解析”按钮后，会在屏幕中列表显示 RSS 新闻，如图 18-3 所示。单击某条新闻后，会显示此新闻的详情，如图 18-4 所示。



图 18-2　初始效果



图 18-3　RSS 新闻

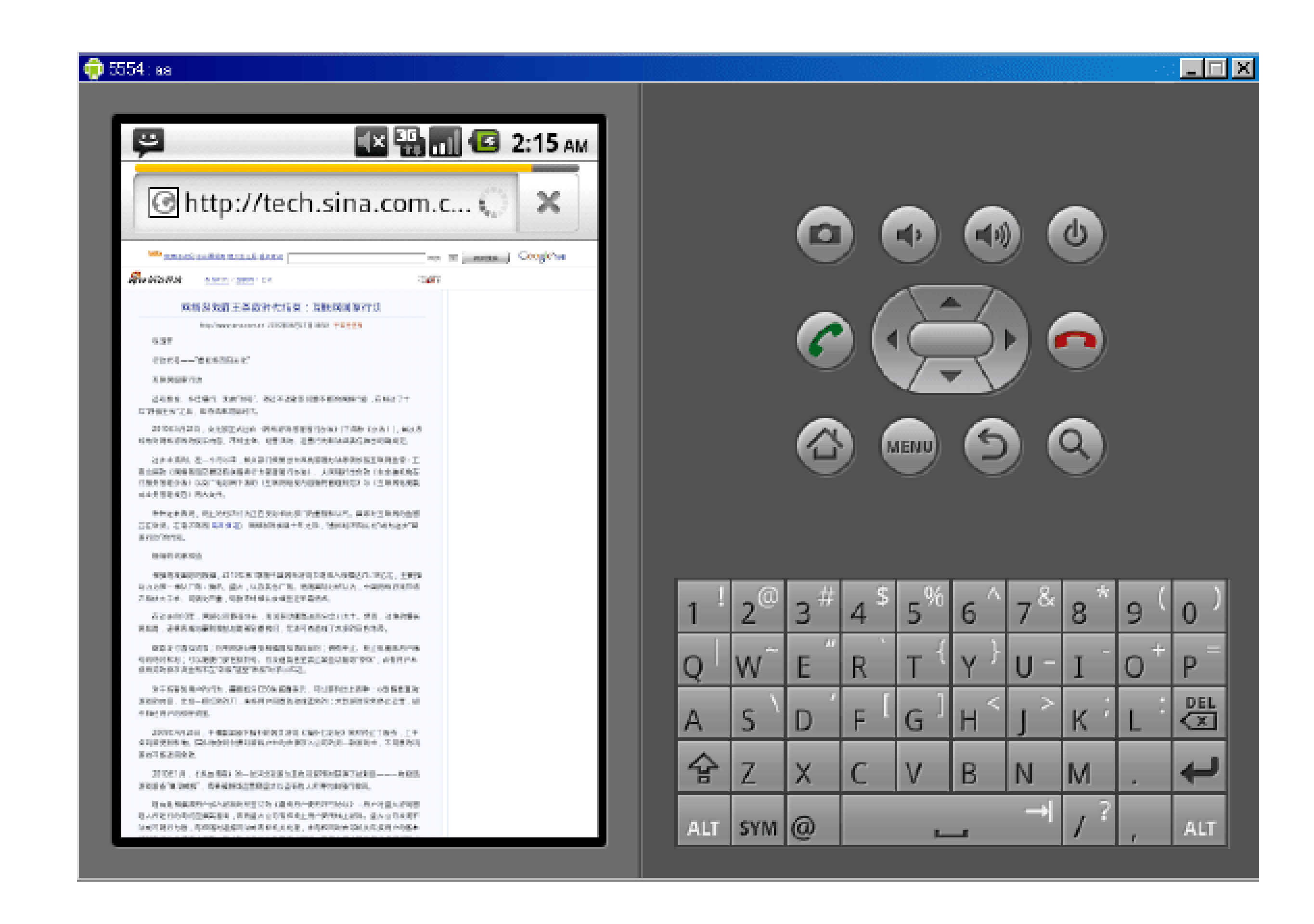


图 18-4　新闻详情

第 19 章　笑傲江湖之个人移动地图

到目前为止，开发内嵌式地图应用的软件是相当困难的，往往还需要支付很高的地图厂商版权费用，加之手机上 GPS 功能的不完善，导致很多可以基于当前位置来开发功能软件的少之又少，Android 提供的地图(Map)功能能够很好地解决上述问题。在本章内容中，将详细介绍 Map 地图在 Android 中的应用，并通过一个综合实例的实现过程来讲解具体实现流程。

19.1　我 的 分 析

我的目的是实现需要的目标定位处理，即当我设置一个目标后，可以在后台启动一个 Service，能够定时读取 GPS 数据以获得用户目前所在的位置信息，并将其保存在数据库中。用户也可以选择其他目标信息，同时能够将这些轨迹显示在 Map 地图上。本项目的具体实现流程如图 19-1 所示。

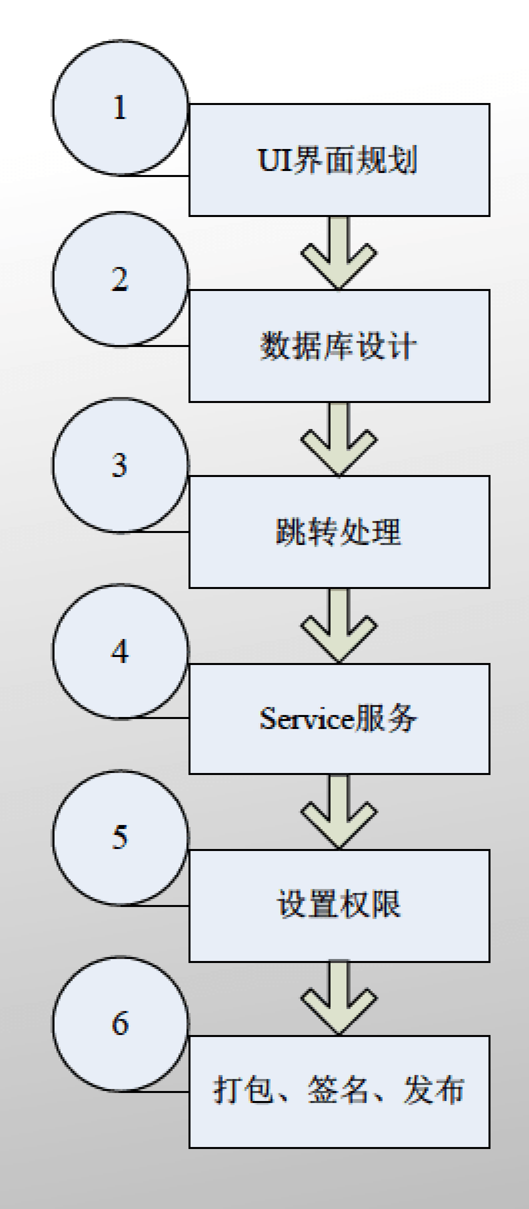


图 19-1　实现流程

19.1.1　规划 UI 界面

分析完系统的运作流程以后，接下来开始规划系统的 UI 界面。根据系统需求，UI 界面结构如图 19-2 所示。

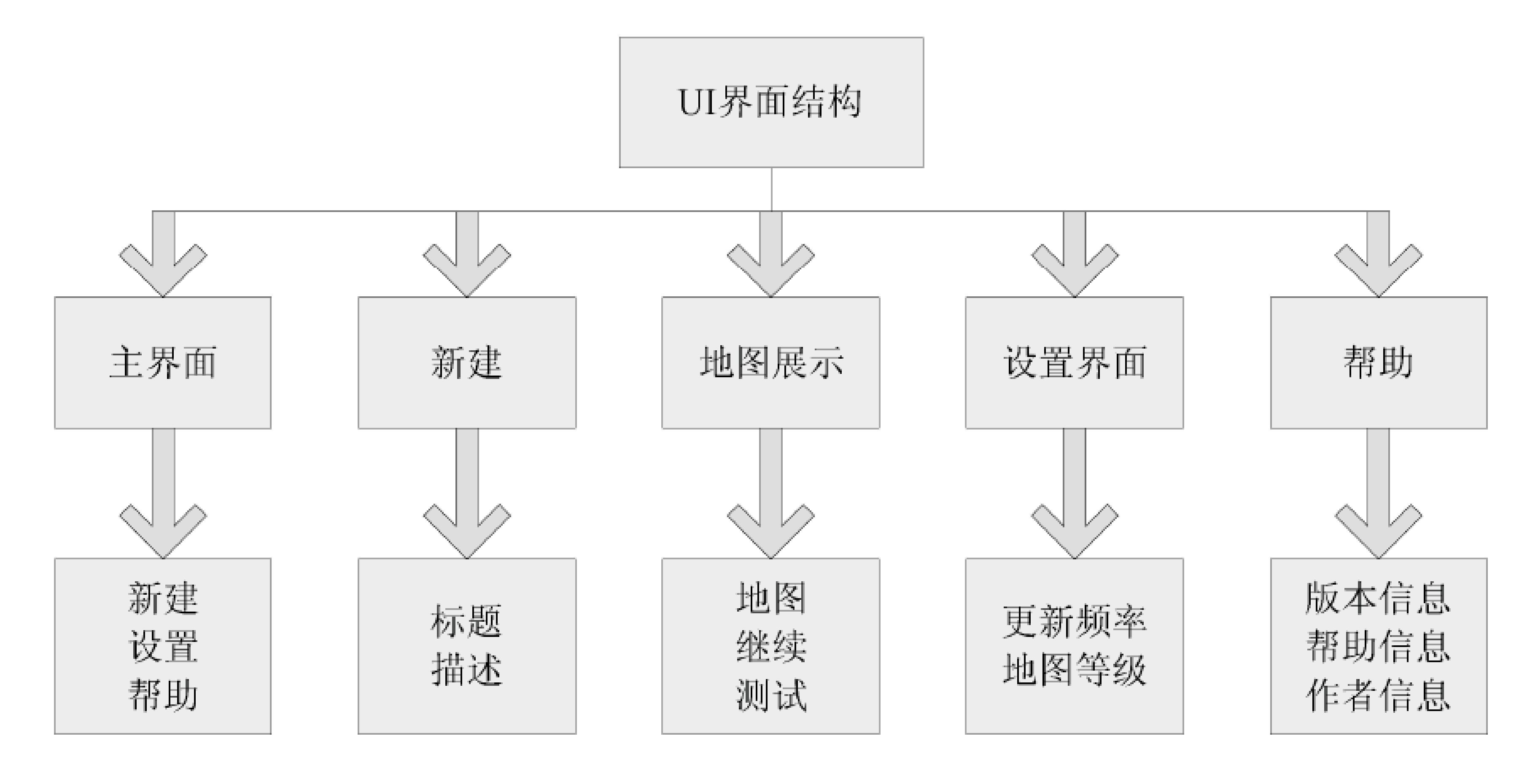


图 19-2　UI 界面结构

19.1.2　数据存储设计

数据存储既可以通过文件系统实现，也可以通过专用数据库工具实现。但是为了保证系统的日常维护工作，本项目使用最常见的数据库工具方式——SQlite。

根据前面介绍的系统需求分析，本系统用到了三类数据：一种是目标名；一种是每次追踪时的位置信息；另外一种是配置信息。在本系统中我设计了两个表来存储数据，具体说明如下。

表 Tracks 用于存储目标信息，具体结构如表 19-1 所示。

表 19-1　Tracks 结构

属　性	类　型	说　明
id	INTEGER	主键
name	TEXT	名
desc	TEXT	说明
distance	LONG	距离
tracked_time	LONG	时间
locates_count	INTEGER	点数
created_at	INTEGER	创建时间
update_at	INTEGER	更新时间
avg_speed	LONG	平均速度
max_speed	LONG	最大速度

表 Locats 用于存储目标的位置信息，具体结构如表 19-2 所示。

表 19-2 Locats 结构

属 性	类 型	说 明
id	INTEGER	主键
track_id	INTEGER	跟踪的目标 ID
longitude	TEXT	纬度
latiude	TEXT	经度
altitude	TEXT	偏差
created_at	INTEGER	创建时间

19.2 具 体 实 现

从本节内容开始，将详细介绍本项目的具体实现过程。希望各位读者仔细阅读，注意细节，争取做到一步到位，这样在实际项目开发中就会有如鱼得水的感觉。

19.2.1 新建工程

打开 Eclipse，依次选择 File | New | Android Project 菜单命令，新建一个名为 map 的工程文件，如图 19-3 所示。

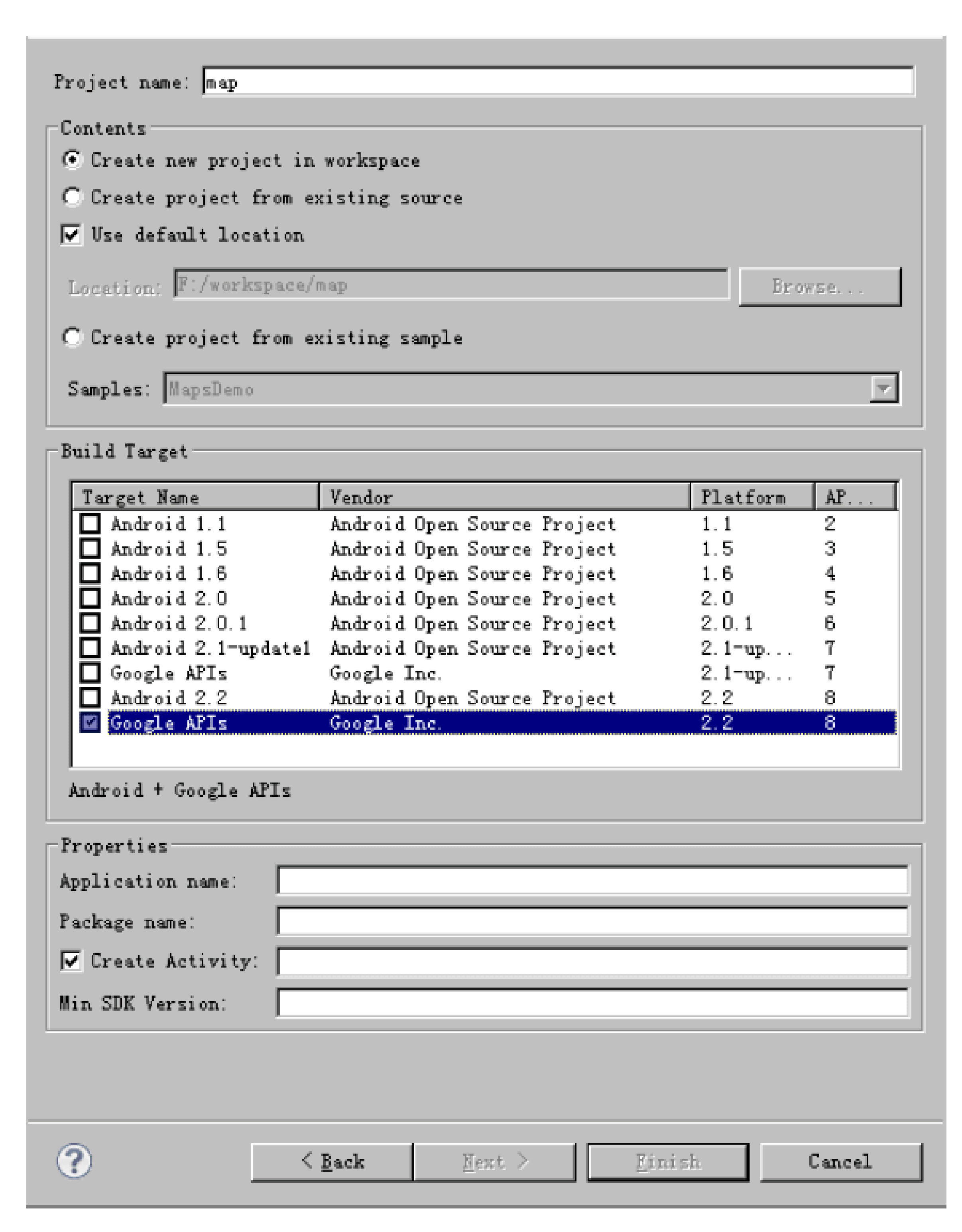


图 19-3 新建工程

19.2.2 主界面

主界面即项目执行后首先显示的界面，创建主界面的具体流程如下。

(1) 编写主布局文件 main.xml。具体代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/title"
   />

    <ListView android:id="@id/android:list"
        android:layout width="fill parent"
        android:layout_height="wrap_content"
        android:drawSelectorOnTop="false" />

    <TextView android:id="@+id/android:empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/start" />
</LinearLayout>
```

(2) 编写一个“历史记录”的列表信息，只显示系统数据库内的前 10 条数据，具体功能是在文件 string.xml 中实现的。具体代码如下：

```
   <string name="title">历史记录:</string>
   <string name="app_name">aaa</string>
   <string name="app_title">bbbb</string>
   <string name="menu_main">主页</string>
   <string name="menu_new">新建</string>
   <string name="menu_con">继续</string>
   <string name="menu_del">删除</string>
   <string name="menu_setting">设置</string>
   <string name="menu_helps">帮助</string>
   <string name="menu_back">返回</string>
   <string name="menu_exit">退出</string>
```

(3) 编写 onCreate 方法。具体代码如下：

```
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
       render_tracks();
   }
```

在上述代码中，需要将以往的历史记录从数据库中读取出来，显示在列表中去，然后使用 render_tracks()方法将数据库的历史记录读取出来，并更新到列表中去。

(4)　在 iTracks.java 中编写实现菜单的代码。主要代码如下：

```
//定义菜单需要的常量
private static final int MENU_NEW = Menu.FIRST + 1;
private static final int MENU CON = MENU NEW + 1;
private static final int MENU SETTING = MENU CON + 1;
private static final int MENU_HELPS = MENU_SETTING + 1;
private static final int MENU_EXIT = MENU_HELPS + 1;
// 初始化菜单
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    Log.d(TAG, "onCreateOptionsMenu");
    super.onCreateOptionsMenu(menu);
    menu.add(0, MENU_NEW, 0, R.string.menu_new).setIcon(
            R.drawable.new_track).setAlphabeticShortcut('N');
    menu.add(0, MENU_CON, 0, R.string.menu_con).setIcon(
            R.drawable.con_track).setAlphabeticShortcut('C');
    menu.add(0, MENU SETTING, 0, R.string.menu setting).setIcon(
            R.drawable.setting).setAlphabeticShortcut('S');
    menu.add(0, MENU_HELPS, 0, R.string.menu_helps).setIcon(
            R.drawable.helps).setAlphabeticShortcut('H');
    menu.add(0, MENU_EXIT, 0, R.string.menu_exit).setIcon(
            R.drawable.exit).setAlphabeticShortcut('E');
    return true;
}

// 当一个菜单被选中的时候调用
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    Intent intent = new Intent();
    switch (item.getItemId()) {
    case MENU_NEW:
        intent.setClass(iTracks.this, NewTrack.class);
        startActivity(intent);
        return true;
    case MENU_CON:
        //TODO: 继续跟踪选择的记录
        conTrackService();
        return true;
    case MENU_SETTING:
        intent.setClass(iTracks.this, Setting.class);
        startActivity(intent);
        return true;
    case MENU_HELPS:
        intent.setClass(iTracks.this, Helps.class);
        startActivity(intent);
        return true;
    case MENU_EXIT:
        finish();
        break;
```

```
        }
        return true;
    }
```

通过上述代码，创建了菜单框架和菜单被选中后的响应方法。至此，主界面的主要代码介绍完毕，执行后的效果如图 19-4 所示。



图 19-4　主界面

19.2.3　新建界面

在图 19-4 中单击“新建”按钮后，会弹出新建目标记录界面。本模块的具体实现流程如下。

(1)　编写布局文件 new_track.xml，分别用 TextView 来显示提示信息，用 EditText 来接收用户的输入。

(2)　编写处理文件 NewTrack.java，主要代码如下：

```
    protected void onStop(){
        super.onStop();
        Log.d(TAG, "onStop");
        mDbHelper.close();
    }
    private void findViews() {
        Log.d(TAG, "find Views");
        field_new_name = (EditText) findViewById(R.id.new_name);
        field_new_desc = (EditText) findViewById(R.id.new_desc);
        button_new = (Button) findViewById(R.id.new_submit);
    }
    // Listen for button clicks
    private void setListensers() {
        Log.d(TAG, "set Listensers");
        button_new.setOnClickListener(new_track);
    }
    private Button.OnClickListener new_track = new Button.OnClickListener() {
        public void onClick(View v) {
            Log.d(TAG, "onClick new_track..");
            try {
                String name = (field_new_name.getText().toString());
                String desc = (field_new_desc.getText().toString());
                if (name.equals("")) {
```

```
                    Toast.makeText(NewTrack.this,
                            getString(R.string.new_name_null),
                            Toast.LENGTH_SHORT).show();
                } else {
                    // TODO 调用存储接口保存到数据库并启动 service
                    Long row_id = mDbHelper.createTrack(name, desc);
                    Log.d(TAG, "row_id="+row_id);
                    Intent intent = new Intent();
                    intent.setClass(NewTrack.this, ShowTrack.class);
                    intent.putExtra(TrackDbAdapter.KEY_ROWID, row_id);
                    intent.putExtra(TrackDbAdapter.NAME, name);
                    intent.putExtra(TrackDbAdapter.DESC, desc);
                    startActivity(intent);
                }
            } catch (Exception err) {
                Log.e(TAG, "error: " + err.toString());
                Toast.makeText(NewTrack.this, getString(R.string.new_fail),
                        Toast.LENGTH_SHORT).show();
            }
        }
    };
}
```

在上述代码中，首先在方法 onCreate()中设置了其关联的 layout；然后调用 findViewsByid()来获取名字、EditText 组件，以及提交按钮；最后，定义了一个 Button.OnClickListener new_track 对象，实现其 onClick()方法。

至此，本模块的主要功能设计完毕，执行后的新建界面如图 19-5 所示。

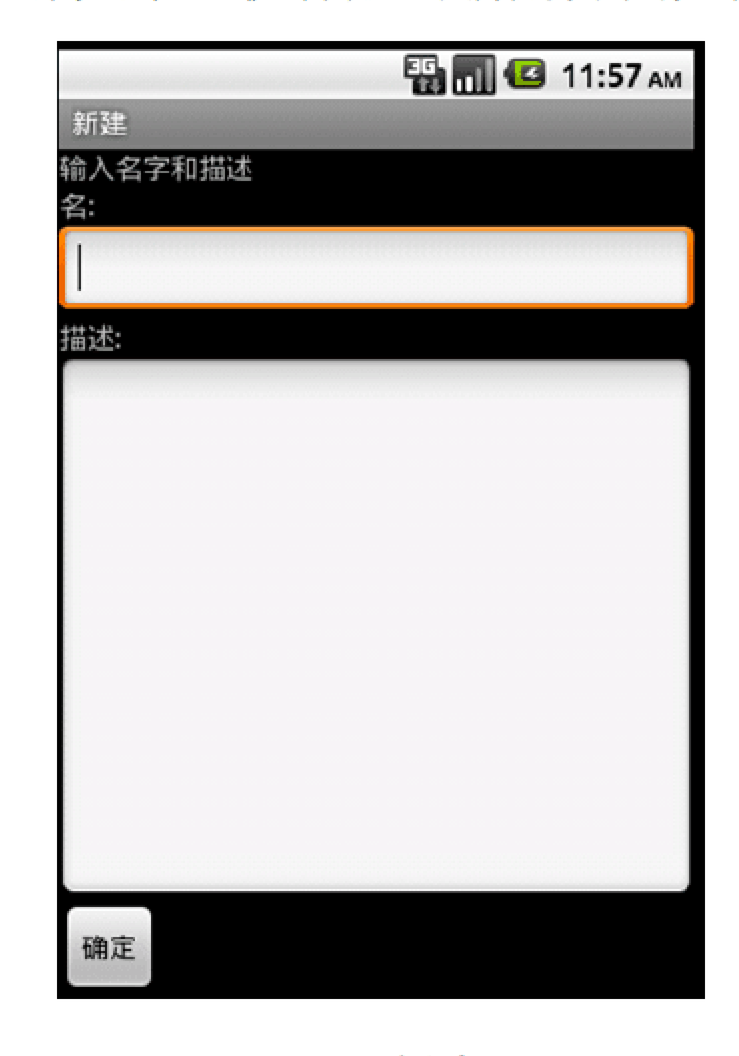


图 19-5　新建界面

19.2.4　设置界面

当在图 19-4 中单击“设置”按钮后，会弹出系统设置界面。本模块的具体实现流程如下：

(1)　编写布局文件 setting.xml，通过 Spinner 组件实现一个供用户使用的下拉菜单。

(2) 编写处理文件 Setting.java，主要代码如下：

```
private void findViews() {
    Log.d(TAG, "find Views");
    button_setting_submit = (Button) findViewById(R.id.setting_submit);
    field_setting_gps = (Spinner) findViewById(R.id.setting_gps);
   ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
          this, R.array.gps, android.R.layout.simple_spinner_item);
   adapter.setDropDownViewResource(android.R.layout.simple_
   spinner_dropdown_item);
   field_setting_gps.setAdapter(adapter);
    field_setting_map_level = (Spinner) findViewById(R.id.setting_map_level);
   ArrayAdapter<CharSequence> adapter2 = ArrayAdapter.createFromResource(
          this, R.array.map, android.R.layout.simple_spinner_item);
   adapter2.setDropDownViewResource(android.R.layout.simple_spinner_
   dropdown_item);
   field_setting_map_level.setAdapter(adapter2);
}

// Listen for button clicks
private void setListensers() {
    Log.d(TAG, "set Listensers");
    button_setting_submit.setOnClickListener(setting_submit);
}
private Button.OnClickListener setting_submit = new Button.OnClickListener() {
    public void onClick(View v) {
        Log.d(TAG, "onClick new_track..");
        try {
            String gps = (field_setting_gps.getSelectedItem().toString());
            String map = (field_setting_map_level.getSelectedItem().toString());
            if (gps.equals("") || map.equals("")) {
                Toast.makeText(Setting.this,
                        getString(R.string.setting_null),
                        Toast.LENGTH_SHORT).show();
            } else {
                //保存设定
                storePrefs();
                Toast.makeText(Setting.this,
                        getString(R.string.setting_ok),
                        Toast.LENGTH_SHORT).show();
                //跳转到主界面
                Intent intent = new Intent();
                intent.setClass(Setting.this, iTracks.class);
                startActivity(intent);
            }
        } catch (Exception err) {
          Log.e(TAG, "error: " + err.toString());
          Toast.makeText(Setting.this, getString(R.string.setting_fail),
                    Toast.LENGTH_SHORT).show();
        }
```

```
        }
    };
    private void restorePrefs() {
        SharedPreferences settings = getSharedPreferences(SETTING_INFOS, 0);
        int setting_gps_p = settings.getInt(SETTING_GPS_POSITON, 0);
        int setting_map_level_p = settings.getInt(SETTING_MAP_POSITON, 0);
        Log.d(TAG, "restorePrefs: setting_gps= "+ setting_gps_p + ",setting_
        map_level=" + setting_map_level_p);
        if (setting_gps_p != 0 && setting_map_level_p != 0) {
            field_setting_gps.setSelection(setting_gps_p);
            field_setting_map_level.setSelection(setting_map_level_p);
            button_setting_submit.requestFocus();
        }else if(setting_gps_p != 0 ){
            field_setting_gps.setSelection(setting_gps_p);
            field_setting_map_level.requestFocus();
        }else if(setting_map_level_p != 0){
            field_setting_map_level.setSelection(setting_map_level_p);
            field_setting_gps.requestFocus();
        }else{
            field_setting_gps.requestFocus();
        }
    }
    @Override
    protected void onStop(){
        super.onStop();
        Log.d(TAG, "save setting infos");
        // Save user preferences. We need an Editor object to
        // make changes. All objects are from android.context.Context
        storePrefs();
    }

    //保存个人设置
    private void storePrefs() {
        Log.d(TAG, "storePrefs setting infos");
        SharedPreferences settings = getSharedPreferences(SETTING_INFOS, 0);
        settings.edit()
            .putString(SETTING_GPS, field_setting_gps.getSelectedItem(). toString())
           .putString(SETTING_MAP, field_setting_map_level.getSelectedItem().
            toString()).putInt(SETTING_GPS_POSITON,
            field_setting_gps.getSelectedItemPosition())
            .putInt(SETTING_MAP_POSITON,
            field_setting_map_level.getSelectedItemPosition())
            .commit();
    }

    // 初始化菜单
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        menu.add(0, MENU_MAIN, 0, R.string.menu_main).setIcon(
```

```
                R.drawable.icon).setAlphabeticShortcut('M');
        menu.add(0, MENU_NEW, 0, R.string.menu_new).setIcon(
                R.drawable.new_track).setAlphabeticShortcut('N');
        menu.add(0, MENU_BACK, 0, R.string.menu_back).setIcon(
                R.drawable.back).setAlphabeticShortcut('E');
        return true;
    }
    // 当一个菜单被选中的时候调用
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        Intent intent = new Intent();
        switch (item.getItemId()) {
        case MENU_NEW:
            intent.setClass(Setting.this, NewTrack.class);
            startActivity(intent);
            return true;
        case MENU_MAIN:
            intent.setClass(Setting.this, iTracks.class);
            startActivity(intent);
            return true;
        case MENU_BACK:
            finish();
            break;
        }
        return true;
    }

}
```

上述代码的具体实现流程如下。

第 1 步：声明需要的变量，并在 onCreat 中绑定 setting.xml 为其布局模板。

第 2 步：使用 setContentView()设定其对应的布局文件 setting.xml，使用 setTitle()设定其标题；进一步调用 findViews()查询到需要的操作组件，并调用 setListensers()给按钮设定单击监听器；最后调用 restorePrefs()将默认值或用户的历史选择值显示出来。

第 3 步：用 findViews()找到需要用的组件。

其中下拉框中的内容是预先设置好的，定义在 array.xml 中，具体代码请参考本书光盘中的源文件。至此，本模块的主要代码介绍完毕，执行后的设置界面效果如图 19-6 所示。



图 19-6　设置界面

19.2.5　帮助界面

当在图 19-4 中单击“帮助”按钮后，会弹出系统默认的帮助界面。本模块的具体实现流程如下。

(1)　编写布局文件 helps.xml，通过 TextView 显示各条帮助信息。

(2)　编写处理文件 helps.java，主要代码如下：

```
public class Helps extends Activity {
    //定义菜单需要的常量
    private static final int MENU_MAIN = Menu.FIRST + 1;
    private static final int MENU_NEW = MENU_MAIN + 1;
    private static final int MENU_BACK = MENU_NEW + 1;;
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.helps);
    setTitle(R.string.menu_helps);
    }
    // 初始化菜单
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        menu.add(0, MENU_MAIN, 0, R.string.menu_main).setIcon(
                R.drawable.icon).setAlphabeticShortcut('M');
        menu.add(0, MENU_NEW, 0, R.string.menu_new).setIcon(
                R.drawable.new_track).setAlphabeticShortcut('N');
        menu.add(0, MENU_BACK, 0, R.string.menu_back).setIcon(
                R.drawable.back).setAlphabeticShortcut('E');
        return true;
    }

    // 当一个菜单被选中的时候调用
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        Intent intent = new Intent();
        switch (item.getItemId()) {
        case MENU_NEW:
            intent.setClass(Helps.this, NewTrack.class);
            startActivity(intent);
            return true;
        case MENU_MAIN:
            intent.setClass(Helps.this, iTracks.class);
            startActivity(intent);
            return true;
        case MENU_BACK:
           finish();
            break;
        }
        return true;
    }
}
```

在上述代码中，首先在 onCreate()方法中设定其对应的布局文件为 helps.xml，然后添加菜单和菜单对应的功能。至此，本模块的主要代码介绍完毕，执行后的帮助界面如图 19-7 所示。

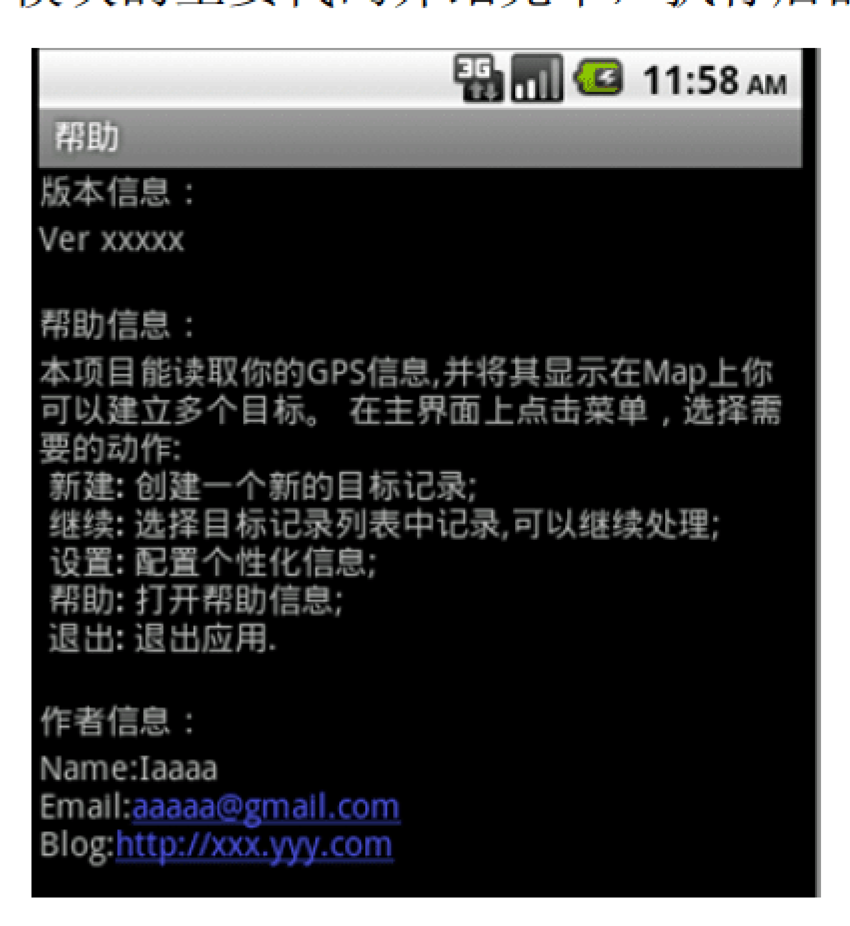


图 19-7 帮助界面

19.2.6 地图界面

前面介绍的都是主菜单中的选项，下面开始讲解比较复杂的功能：将地图在 Android 手机中显示。前面已经讲解了 com.google.android.maps 的基本知识，通过其中的 MapView 即可方便地实现编程工作。在申请之后，即可进行编码工作。下面开始讲解具体的编码过程。

(1) 编写布局文件 show_track.xml，通过 MapView 组件来显示地图，并通过设置的按钮来控制地图，例如放大、缩小、移动和模式的转换。

(2) 编写处理文件 ShowTrack.java。主要代码如下：

```
private void startTrackService() {
    Intent i = new Intent("com.iceskysl.iTracks.START TRACK SERVICE");
    i.putExtra(LocateDbAdapter.TRACKID, track_id);
   startService(i);
}

private void stopTrackService() {
   stopService(new Intent("com.iceskysl.iTracks.START_TRACK_SERVICE"));

}
private void paintLocates() {
    mlcDbHelper = new LocateDbAdapter(this);
    mlcDbHelper.open();
    Cursor mLocatesCursor = mlcDbHelper.getTrackAllLocates(track id);
    startManagingCursor(mLocatesCursor);
    Resources resources = getResources();
    Overlay overlays = new LocateOverLay(resources
            .getDrawable(R.drawable.icon), mLocatesCursor);
    mMapView.getOverlays().add(overlays);
    mlcDbHelper.close();
}
private void revArgs() {
```

```
        Log.d(TAG, "revArgs.");
        Bundle extras = getIntent().getExtras();
        if (extras != null) {
            String name = extras.getString(TrackDbAdapter.NAME);
            //String desc = extras.getString(TrackDbAdapter.DESC);
            rowId = extras.getLong(TrackDbAdapter.KEY_ROWID);
            track_id = rowId.intValue();
            Log.d(TAG, "rowId=" + rowId);
            if (name != null) {
                setTitle(name);
            }
        }
    }
    protected boolean isRouteDisplayed() {
        return false;
    }

    private void findViews() {
        Log.d(TAG, "find Views");//获取视图中的组件
        mMapView = (MapView) findViewById(R.id.mv);
        mc = mMapView.getController();
        //地图设置
        SharedPreferences settings = getSharedPreferences(Setting.SETTING_INFOS, 0);
        String setting_gps = settings.getString(Setting.SETTING_MAP, "10");
        mc.setZoom(Integer.parseInt(setting gps));
        //设置向东移动按钮
        mPanE = (Button) findViewById(R.id.sat);
        mPanE.setOnClickListener(new OnClickListener() {
            // @Override
            public void onClick(View arg0) {
                panEast();
            }
        });
        //缩小按钮
        mZin = (Button) findViewById(R.id.zin);
        mZin.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0) {
                zoomIn();
            }
        });
        //放大按钮
        mZout = (Button) findViewById(R.id.zout);
        mZout.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0) {
                zoomOut();
            }
        });
        mPanN = (Button) findViewById(R.id.pann);
        mPanN.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0) {
                panNorth();
            }
```




```
        });
        mPanE = (Button) findViewById(R.id.pane);
        mPanE.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0) {
                panEast();
            }
        });
        //向西移动按钮
        mPanW = (Button) findViewById(R.id.panw);
        mPanW.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0) {
                panWest();
            }
        });
        //向南移动按钮
        mPanS = (Button) findViewById(R.id.pans);
        mPanS.setOnClickListener(new OnClickListener() {
            // @Override
            public void onClick(View arg0) {
                panSouth();
            }
        });
        mGps = (Button) findViewById(R.id.gps);
        mGps.setOnClickListener(new OnClickListener() {
            // @Override
            public void onClick(View arg0) {
                centerOnGPSPosition();
            }
        });
        //卫星视图开关
        mSat = (Button) findViewById(R.id.sat);
        mSat.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0) {
                toggleSatellite();
            }
        });
        //交通地图
        mTraffic = (Button) findViewById(R.id.traffic);
        mTraffic.setOnClickListener(new OnClickListener() {
            // @Override
            public void onClick(View arg0) {
                toggleTraffic();
            }
        });
        //街景地图
        mStreetview = (Button) findViewById(R.id.streetview);
        mStreetview.setOnClickListener(new OnClickListener() {
            // @Override
            public void onClick(View arg0) {
                toggleStreetView();
            }
        });
```

```
        //使用位置管理器获取 GPS 位置的变化信息
        lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        locationListener = new MyLocationListener();
        lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
                locationListener);
    }
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        Log.d(TAG, "onKeyDown");
        if (keyCode == KeyEvent.KEYCODE_DPAD_LEFT) {
            panWest();
            return true;
        } else if (keyCode == KeyEvent.KEYCODE_DPAD_RIGHT) {
            panEast();
            return true;
        } else if (keyCode == KeyEvent.KEYCODE_DPAD_UP) {
            panNorth();
            return true;
        } else if (keyCode == KeyEvent.KEYCODE_DPAD_DOWN) {
            panSouth();
            return true;
        }
        return false;
    }
    //下面是 4 个方向移动的处理方法
    public void panWest() {
        GeoPoint pt = new GeoPoint(mMapView.getMapCenter().getLatitudeE6(),
                mMapView.getMapCenter().getLongitudeE6()
                        - mMapView.getLongitudeSpan() / 4);
        mc.setCenter(pt);
    }
    public void panEast() {
        GeoPoint pt = new GeoPoint(mMapView.getMapCenter().getLatitudeE6(),
                mMapView.getMapCenter().getLongitudeE6()
                        + mMapView.getLongitudeSpan() / 4);
        mc.setCenter(pt);
    }

    public void panNorth() {
        GeoPoint pt = new GeoPoint(mMapView.getMapCenter().getLatitudeE6()
                + mMapView.getLatitudeSpan() / 4, mMapView.getMapCenter()
                .getLongitudeE6());
        mc.setCenter(pt);
    }

    public void panSouth() {
        GeoPoint pt = new GeoPoint(mMapView.getMapCenter().getLatitudeE6()
                - mMapView.getLatitudeSpan() / 4, mMapView.getMapCenter()
                .getLongitudeE6());
        mc.setCenter(pt);
    }
    //下面是放大和缩小的具体处理方法
    public void zoomIn() {
```

```
        mc.zoomIn();
    }

    public void zoomOut() {
        mc.zoomOut();
    }
    //下面是交通、街景、卫星地图的开关处理方法
    public void toggleSatellite() {
        mMapView.setSatellite(true);
        mMapView.setStreetView(false);
        mMapView.setTraffic(false);
    }
    public void toggleTraffic() {
        mMapView.setTraffic(true);
        mMapView.setSatellite(false);
        mMapView.setStreetView(false);
    }

    public void toggleStreetView() {
        mMapView.setStreetView(true);
        mMapView.setSatellite(false);
        mMapView.setTraffic(false);
    }
    //将地图定位到当前 GPS 指定的位置
    private void centerOnGPSPosition() {
        Log.d(TAG, "centerOnGPSPosition");
        String provider = "gps";
        LocationManager lm = (LocationManager) getSystemService(Context. LOCATION_
        SERVICE);

        Location loc = lm.getLastKnownLocation(provider);
        loc = lm.getLastKnownLocation(provider);
        mDefPoint = new GeoPoint((int) (loc.getLatitude() * 1000000),
                (int) (loc.getLongitude() * 1000000));
        mDefCaption = "I'm Here.";
        mc.animateTo(mDefPoint);
        mc.setCenter(mDefPoint);
        // show Overlay on map.
        MyOverlay mo = new MyOverlay();
        mo.onTap(mDefPoint, mMapView);
        mMapView.getOverlays().add(mo);
    }
    protected class MyOverlay extends Overlay {
        @Override
        public void draw(Canvas canvas, MapView mv, boolean shadow) {
            Log.d(TAG, "MyOverlay::darw..mDefCaption=" + mDefCaption);
            super.draw(canvas, mv, shadow);

            if (mDefCaption.length() == 0) {
                return;
            }
            Paint p = new Paint();
```

```
            int[] scoords = new int[2];
            int sz = 5;
            // 转换为屏幕上的位置
            Point myScreenCoords = new Point();
            mMapView.getProjection().toPixels(mDefPoint, myScreenCoords);
            // mMapView.set
            // mv.getPointXY(mDefPoint, scoords);
            // 在屏幕上绘制点
            scoords[0] = myScreenCoords.x;
            scoords[1] = myScreenCoords.y;
            p.setTextSize(14);
            p.setAntiAlias(true);
            int sw = (int) (p.measureText(mDefCaption) + 0.5f);
            int sh = 25;
            int sx = scoords[0] - sw / 2 - 5;
            int sy = scoords[1] - sh - sz - 2;
            RectF rec = new RectF(sx, sy, sx + sw + 10, sy + sh);
            p.setStyle(Style.FILL);
            p.setARGB(128, 255, 0, 0);
            canvas.drawRoundRect(rec, 5, 5, p);
            p.setStyle(Style.STROKE);
            p.setARGB(255, 255, 255, 255);
            canvas.drawRoundRect(rec, 5, 5, p);
            canvas.drawText(mDefCaption, sx + 5, sy + sh - 8, p);
            p.setStyle(Style.FILL);
            p.setARGB(88, 255, 0, 0);
            p.setStrokeWidth(1);
            RectF spot = new RectF(scoords[0] - sz, scoords[1] + sz, scoords[0]
                    + sz, scoords[1] - sz);
            canvas.drawOval(spot, p);
            p.setARGB(255, 255, 0, 0);
            p.setStyle(Style.STROKE);
            canvas.drawCircle(scoords[0], scoords[1], sz, p);
        }
    }

    protected class MyLocationListener implements LocationListener {

        @Override
        public void onLocationChanged(Location loc) {
            Log.d(TAG, "MyLocationListener::onLocationChanged..");
            if (loc != null) {
                Toast.makeText( getBaseContext(), "Location changed : Lat: " +
                loc.getLatitude() + " Lng: " + loc.getLongitude(),
                        Toast.LENGTH_SHORT).show();
                // Set up the overlay controller
                // mOverlayController = mMapView.createOverlayController();
                mDefPoint = new GeoPoint((int) (loc.getLatitude() * 1000000),
                        (int) (loc.getLongitude() * 1000000));
                mc.animateTo(mDefPoint);
                mc.setCenter(mDefPoint);
                // 在地图上显示
```

```
                mDefCaption = "Lat: " + loc.getLatitude() + ",Lng: "
                        + loc.getLongitude();
                MyOverlay mo = new MyOverlay();
                mo.onTap(mDefPoint, mMapView);
                mMapView.getOverlays().add(mo);
                // //////////
                //if(mlcDbHelper == null){
                //  mlcDbHelper.open();
                //}
                //mlcDbHelper.createLocate(track_id, loc.getLongitude(),
                loc.getLatitude(), loc.getAltitude());
            }
        }
        @Override
        public void onProviderDisabled(String provider) {
            Toast.makeText(
                    getBaseContext(),
                    "ProviderDisabled.",
                    Toast.LENGTH_SHORT).show();     }
        @Override
        public void onProviderEnabled(String provider) {
            Toast.makeText(
                    getBaseContext(),
                    "ProviderEnabled,provider:"+provider,
                    Toast.LENGTH_SHORT).show();     }
        @Override
        public void onStatusChanged(String provider, int status, Bundle extras)
{
            // TODO Auto-generated method stub
        }
    }
    // 初始化菜单
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        menu.add(0, MENU_CON, 0, R.string.menu_con).setIcon(
                R.drawable.con_track).setAlphabeticShortcut('C');
        menu.add(0, MENU_DEL, 0, R.string.menu_del).setIcon(R.drawable.delete)
                .setAlphabeticShortcut('D');
        menu.add(0, MENU_NEW, 0, R.string.menu_new).setIcon(
                R.drawable.new_track).setAlphabeticShortcut('N');
        menu.add(0, MENU_MAIN, 0, R.string.menu_main).setIcon(R.drawable.icon)
                .setAlphabeticShortcut('M');
        return true;
    }
    // 当一个菜单被选中的时候调用
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        Intent intent = new Intent();
        switch (item.getItemId()) {
        case MENU_NEW:
            intent.setClass(ShowTrack.this, NewTrack.class);
```

```
            startActivity(intent);
            return true;
        case MENU_CON:
            // TODO: 继续跟踪选择的记录
            startTrackService();
            return true;
        case MENU_DEL:
             mDbHelper = new TrackDbAdapter(this);
             mDbHelper.open();
            if (mDbHelper.deleteTrack(rowId)) {
                mDbHelper.close();
                intent.setClass(ShowTrack.this, iTracks.class);
                startActivity(intent);
            }else{
                mDbHelper.close();
            }
            return true;
        case MENU_MAIN:
            intent.setClass(ShowTrack.this, iTracks.class);
            startActivity(intent);
            break;
        }
        return true;
    }
    @Override
    protected void onStop() {
        super.onStop();
        Log.d(TAG, "onStop");
        // mDbHelper.close();
        //mlcDbHelper.close();
    }
    @Override
   public void onDestroy() {
        Log.d(TAG, "onDestroy.");
       super.onDestroy();
       stopTrackService();
       }
}
```

上述代码的具体实现流程如下。

第 1 步：通过 findViews 来确定要使用的控件，并绑定需要响应的事件。

第 2 步：通过 findViews 实现对地图的处理，首先获取布局中的 MapView，使用 getController 得到一个 MapController，然后注册一个基于 locationListener 的 MyLocationListener。

第 3 步：实现处理按钮的处理方法代码，具体原理比较简单，即先获取地图中心，然后向四个方向移动 1/4 距离。

第 4 步：单击“GPS”按钮的响应方法 centerOnGPSPosition，即将地图定位到当前 GPS 指定的位置。

第 5 步：通过 Overylay 抽象类重载实现 draw()方法。

执行后的地图显示效果如图 19-8 所示。



图 19-8　地图展示界面

19.2.7　数据存取

在前面介绍的系统需求分析中，系统要求将每次跟踪的目标位置保存在数据库中，并且每次改变后都要保存起来。本项目的个性化配置信息保存在 SharedPreferences 中，在此需要将存取的数据放在数据库中。在 Android 中，存取数据的方法有两种，一种是通过 help 类继承 SQLiteDatabase 相关类绑定 SQL；另外一种是使用 ContentProvideer 进行封装。

1. 创建数据库

本项目需要同时操作数据库中的两个表，在此先在文件 DbAdapter.java 中创建一个名为 DbAdapter 的类。主要实现代码如下：

```
public class DbAdapter {
    private static final String TAG = "DbAdapter";
    private static final String DATABASE_NAME = "iTracks.db";
    private static final int DATABASE_VERSION = 1;

    public class DatabaseHelper extends SQLiteOpenHelper {
        public DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }
        @Override
        public void onCreate(SQLiteDatabase db) {
            String tracks sql = "CREATE TABLE " + TrackDbAdapter.TABLE NAME + " ("
            + TrackDbAdapter.ID + " INTEGER primary key autoincrement, "
            + TrackDbAdapter.NAME   + " text not null, "
            + TrackDbAdapter.DESC + " text ,"
            + TrackDbAdapter.DIST + " LONG ,"
            + TrackDbAdapter.TRACKEDTIME + " LONG ,"
            + TrackDbAdapter.LOCATE_COUNT + " INTEGER, "
            + TrackDbAdapter.CREATED + " text, "
            + TrackDbAdapter.AVGSPEED + " LONG, "
            + TrackDbAdapter.MAXSPEED + " LONG ,"
```

```
                + TrackDbAdapter.UPDATED + " text "
                + ");";
                Log.i(TAG, tracks_sql);
                db.execSQL(tracks_sql);
                String locats_sql = "CREATE TABLE " + LocateDbAdapter.TABLE_NAME + " ("
                + LocateDbAdapter.ID    + " INTEGER primary key autoincrement, "
                + LocateDbAdapter.TRACKID   + " INTEGER not null, "
                + LocateDbAdapter.LON + " DOUBLE ,"
                + LocateDbAdapter.LAT + " DOUBLE ,"
                + LocateDbAdapter.ALT + " DOUBLE ,"
                + LocateDbAdapter.CREATED + " text "
                + ");";
                Log.i(TAG, locats_sql);
                db.execSQL(locats_sql);
            }
            @Override
            public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                db.execSQL("DROP TABLE IF EXISTS " + LocateDbAdapter.TABLE_NAME + ";");
                db.execSQL("DROP TABLE IF EXISTS " + TrackDbAdapter.TABLE_NAME + ";");
                onCreate(db);
            }
        }

}
```

在上述代码中，重新定义了 SQLiteOpenHelper 的 onCreate()方法和 onUpgrade()方法，通过这两个方法实现了创建和升级数据库的脚本。

2. 数据库操作

本功能实现了对两个表操作的封装处理，因为共用了同一个数据库，所以只需从前面创建的 DbAdapter 类中继续继承即可，在此继承了两个类：TrackDbAdapter 和 LocateDbAdapter。通过对这两个类的封装，实现了对数据表的操作。数据库操作的具体流程如下。

(1) TrackDbAdapter 类是在文件 TrackDbAdapter.java 中定义的。主要实现代码如下：

```
    public TrackDbAdapter(Context ctx) {
        this.mCtx = ctx;
    }
    public TrackDbAdapter open() throws SQLException {
        mDbHelper = new DatabaseHelper(mCtx);
        mDb = mDbHelper.getWritableDatabase();
        return this;
    }
    public void close() {
        mDbHelper.close();
    }
    public Cursor getTrack(long rowId) throws SQLException {
        Cursor mCursor =
        mDb.query(true, TABLE_NAME, new String[] { KEY_ROWID, NAME,
                DESC, CREATED }, KEY_ROWID + "=" + rowId, null, null,
                null, null, null);
        if (mCursor != null) {
```

```
            mCursor.moveToFirst();
        }
        return mCursor;
    }
    public long createTrack(String name, String desc) {
        Log.d(TAG, "createTrack.");
        ContentValues initialValues = new ContentValues();
        initialValues.put(NAME, name);
        initialValues.put(DESC, desc);
        Calendar calendar = Calendar.getInstance();
        String created = calendar.get(Calendar.YEAR) + "-" +calendar.get
        (Calendar.MONTH) + "-" +  calendar.get(Calendar.DAY OF MONTH) + " "
                + calendar.get(Calendar.HOUR OF DAY) + ":"
                + calendar.get(Calendar.MINUTE) + ":"  + calendar.get
                (Calendar.SECOND);
        initialValues.put(CREATED, created);
        initialValues.put(UPDATED, created);
        return mDb.insert(TABLE_NAME, null, initialValues);
    }
    public boolean deleteTrack(long rowId) {
        return mDb.delete(TABLE_NAME, KEY_ROWID + "=" + rowId, null) > 0;
    }

    public Cursor getAllTracks() {
        return mDb.query(TABLE NAME, new String[] { ID, NAME,
                DESC, CREATED }, null, null, null, null, "updated at desc");
    }

    public boolean updateTrack(long rowId, String name, String desc) {
        ContentValues args = new ContentValues();
        args.put(NAME, name);
        args.put(DESC, desc);
        Calendar calendar = Calendar.getInstance();
        String updated = calendar.get(Calendar.YEAR) + "-" +calendar.get
        (Calendar.MONTH) + "-" +  calendar.get(Calendar.DAY_OF_MONTH) + " "
        + calendar.get(Calendar.HOUR_OF_DAY) + ":"
        + calendar.get(Calendar.MINUTE) + ":"  + calendar.get(Calendar.SECOND);
        args.put(UPDATED, updated);
        return mDb.update(TABLE_NAME, args, KEY_ROWID + "=" + rowId, null) > 0;
    }

}
```

在上述代码中，首先声明了一些常量，然后根据需要的操作功能定义具体方法。

(2) LocateDbAdapter 类是在文件 LocateDbAdapter.java 中实现的。主要实现代码如下：

```
    public LocateDbAdapter(Context ctx) {
        this.mCtx = ctx;
    }
    public LocateDbAdapter open() throws SQLException {
        mDbHelper = new DatabaseHelper(mCtx);
        mDb = mDbHelper.getWritableDatabase();
```

```
        return this;
    }
    public void close() {
        mDbHelper.close();
    }
    public Cursor getLocate(long rowId) throws SQLException {
        Cursor mCursor =
        mDb.query(true, TABLE_NAME, new String[] { ID, LON,
                LAT, ALT, CREATED }, ID + "=" + rowId, null, null,
                null, null, null);
        if (mCursor != null) {
            mCursor.moveToFirst();
        }
        return mCursor;
    }

    public long createLocate(int track_id, Double longitude, Double latitude,
    Double altitude) {
        Log.d(TAG, "createLocate.");
        ContentValues initialValues = new ContentValues();
        initialValues.put(TRACKID, track_id);
        initialValues.put(LON, longitude);
        initialValues.put(LAT, latitude);
        initialValues.put(ALT, altitude);
        Calendar calendar = Calendar.getInstance();
       String created = calendar.get(Calendar.YEAR) + "-" +calendar.get(Calendar.MONTH)
        + "-" +  calendar.get(Calendar.DAY_OF_MONTH) + " "
                + calendar.get(Calendar.HOUR_OF_DAY) + ":"
                + calendar.get(Calendar.MINUTE) + ":" + calendar.get(Calendar.SECOND);
        initialValues.put(CREATED, created);
        return mDb.insert(TABLE_NAME, null, initialValues);
    }
    public boolean deleteLocate(long rowId) {
        return mDb.delete(TABLE_NAME, ID + "=" + rowId, null) > 0;
    }

    public Cursor getTrackAllLocates(int trackId) {
        return mDb.query(TABLE_NAME, new String[] { ID,TRACKID, LON,
               LAT, ALT,CREATED }, "track_id=?", new String[] {String.valueOf(trackId)},
                null, null, "created_at asc");
    }
}
```

在上述代码中，也是首先声明了一些常量，然后根据需要的操作功能定义具体方法。

19.2.8　实现 Service 服务

本项目的基本要求是切换一个界面不会影响到对目标的追踪。即使来到另外一个界面，程序也需要在后台进行跟踪和记录，所以需要用到 Service 服务，首先在文件 AndroidManifest.xml 中加入对 Service 的声明。具体代码如下：

```
<service android:name=".Track">
    <intent-filter>
        <action android:name="com.iceskysl.map.START_TRACK_SERVICE" />
        <category android:name="android.intent.category.default" />
    </intent-filter>
</service>
```

通过上述代码添加了一个名为 Track 的 Service，并设定了其名字为“com.iceskysl.map.START_TRACK_SERVICE”，处理文件 Track.java 的主要实现代码如下：

```
    public class Track extends Service {
    private static final String TAG = "Track";

    private LocationManager lm;
    private LocationListener locationListener;
    static LocateDbAdapter mlcDbHelper = null;
    private int track_id;
    @Override
    public IBinder onBind(Intent arg0) {
        Log.d(TAG, "onBind.");
        return null;
    }
    public void onStart(Intent intent, int startId) {
        Log.d(TAG, "onStart.");
       super.onStart(intent, startId);
       startDb();
        Bundle extras = intent.getExtras();
        if (extras != null) {
            track_id = extras.getInt(LocateDbAdapter.TRACKID);
        }
        Log.d(TAG, "track id =" + track id);
        // ---use the LocationManager class to obtain GPS locations---
        lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        locationListener = new MyLocationListener();
        lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
                locationListener);
    }
    private void startDb() {
        if(mlcDbHelper == null){
            mlcDbHelper = new LocateDbAdapter(this);
             mlcDbHelper.open();
        }
    }
    private void stopDb() {
        if(mlcDbHelper != null){
           mlcDbHelper.close();
        }
    }

    public static LocateDbAdapter getDbHelp(){
        return mlcDbHelper;
    }
```

```
    public void onDestroy() {
         Log.d(TAG, "onDestroy.");
        super.onDestroy();
        stopDb();
    }
    protected class MyLocationListener implements LocationListener {
         @Override
         public void onLocationChanged(Location loc) {
             Log.d(TAG, "MyLocationListener::onLocationChanged..");
             if (loc != null) {
                 // //////////
                 if(mlcDbHelper == null){
                     mlcDbHelper.open();
                 }
                 mlcDbHelper.createLocate(track_id,
loc.getLongitude(),loc.getLatitude(), loc.getAltitude());
             }
         }
         @Override
         public void onProviderDisabled(String provider) {
             Toast.makeText(
                     getBaseContext(),
                     "ProviderDisabled.",
                     Toast.LENGTH_SHORT).show();     }
         @Override
         public void onProviderEnabled(String provider) {
             Toast.makeText(
                     getBaseContext(),
                     "ProviderEnabled,provider:"+provider,
                     Toast.LENGTH_SHORT).show();     }
         @Override
         public void onStatusChanged(String provider, int status, Bundle extras)
{
             // TODO Auto-generated method stub
         }
     }
}
```

在上述代码中，Track 继承了 Service 类，然后在 onStart()方法中连接了数据库，接收了参数并设定了监听器，并使用了 MyLocationListener，使在当前位置发生变化(onLoacationChanged)的时候，调用前面数据存储部分已经实现的 mlcDbHelper.createLocate()方法，将位置信息和接收到的参数写入到数据库中。

第 20 章　尘埃落定之游戏

手机游戏开创了一种全新的娱乐方式和应用模式，并随着移动互联网的发展而火爆。我们可以随处可见在大街上、公车上、公园中玩手机游戏的人。因此开发一款 Android 手机游戏很有必要，在本章的内容中，将通过一个具体实例的实现过程，介绍开发一款 Android 游戏的基本流程。

20.1　蓬勃发展的手机游戏

手机游戏开创了一种全新的娱乐方式和应用模式，并随着移动互联网的发展而火爆。2010 年第 3 季度中国手机游戏用户数量突破 1.2 亿，达到了 1.208 亿，环比增长 9.1%，同期手机游戏市场规模达到 9.175 亿。

20.1.1　1.2 亿手机游戏用户

近期，一款名叫“愤怒的小鸟”的手机游戏异常火爆，这款游戏已经出现在多个手机平台上，仅单单在 iOS 平台上的总下载量就已经超过了 1000 万次，显示出巨大的爆发力。

当前国内几大运营商已纷纷采取行动，投建各自的手机游戏产品基地，为拓宽手机游戏市场做足准备。最近，中国电信游戏运营中心在江苏挂牌成立，“爱游戏”产品同步上线，手机游戏在业内掀起一轮小高潮。

与此同时，国内的手机游戏开发商也不断地加大对新款游戏的开发力度，并积极建立更多的非运营商渠道。今年年初，盛大游戏宣布以 8000 万美元的价格全资收购美国游戏分销和内置广告平台 MochiMedia，而 MochiMedia 的 CEO Jameson Hsu 则称，进入盛大麾下之后将会注重社区游戏和手机游戏上的布局；近期宣布将引入手机游戏平台 OpenFeint 的第九城市，7 月初就已宣布与 OpenFeint 的母公司——美国手机游戏及平台公司 Aurora Feint 达成最终投资协议，对其进行战略性少数股权投资。第九城市还透露有望于年内推出多个手机游戏领域项目；几年前已涉足手机游戏业务的腾讯，日前也透露 “看好高端手机的游戏平台”并已成立工作室，专门做高端手机平台的游戏研发和平台研发；近日有消息称，英特尔也将向手机游戏平台 OpenFeint

投资 300 万美元。

另外，在终端生产方面，新的手机型号层出不穷，对各种手机游戏的支持也日趋全面化。同时，手机游戏内容和体验均有较高的提升。

20.1.2　手机游戏业务成淘金点

目前，移动互联网用户在手机网游、手机阅读、移动微博等细分领域的需求表现更为迫切，这使得移动互联网应用服务得以快速发展，移动娱乐等各方面应用表现得更为突出。而对于这些用户而言，手机游戏无疑是移动娱乐的先锋应用，市场前景大好。

随着 3G 的普及以及智能手机的推广，移动互联网在我国已呈现成熟迹象，逐渐成为众厂商觊觎的领域。毫无疑问，智能手机在生活中所扮演的角色越来越重要，甚至成为手机换代的趋势。而在智能手机丰富的用户软件平台上，手机游戏、流媒体、手机动漫等仍将是推动无线增值业务高增长的主要业务。特别是手机游戏业务，一直是 3G 产业中发展最重要的一个淘金点，受到各厂家的狂热追捧。

有分析人士称，“因为智能手机高质量的触摸屏，强大的程序处理器，优化的图像及摄像功能，更大的内存容量，加速器和 GPS 等功能都变得更加标准，所以更有利于提高手机游戏体验。”随着智能手机的快速普及，尤其是 iPhone、Android 和 iPad 带来的一种热潮，都对手机游戏成为先锋应用有很好地促进作用。

20.1.3　任重而道远的现实

虽然整个产业蓬勃发展，但是现实却改变不了。现实是我国手机游戏产业暂处于起步阶段，离“跨越式”发展还存在很大的差距，其进一步发展仍需克服不少难题。首先，短信代收的三次确认影响了手机单机游戏开发商向用户有效收费；其次，手机游戏开发商重视产品数量大于产品质量，而产品从类型和题材上又存在较为严重的同质化现象；再次，虽然手机游戏运营平台呈现多元化的利好趋势，但目前正处在调整期，市场出现观望现象；另外，伴随移动终端和通讯网络的发展，将出现更多的手机娱乐方式，如何给用户全新的体验，让用户选择手机游戏，也是摆在眼前的重大难题。

然而，伴随着智能手机、移动网络的不断升级，手机游戏发展空间也越来越大。从目前市场规模和用户规模来看，手机游戏是一个潜力巨大、尚待发掘的领域，需要在内容、传输渠道、营销、收费等方面加以改进，谋求更广阔的发展出路。

20.2　Java 游戏开发面面观

手机游戏是指运行于手机上的游戏软件。目前用来编写手机游戏软件最多的程序语言是 Java，例如常见的 J2ME。随着科技的发展，现在手机的功能也越来越多，越来越强大。而手机游戏也随之逐渐发展，早已经不是印象中的诸如俄罗斯方块和贪吃蛇之类画面简陋、规则简单的游戏，进而发展到了可以和掌上游戏机媲美，具有很强的娱乐性和交互性的复杂形态了。现实是买一个好的手机就能够满足你所有需求中大部分的游戏娱乐功能了。开发一款典型 Java 游戏的流程如图 20-1 所示，基本过程的具体说明如下。

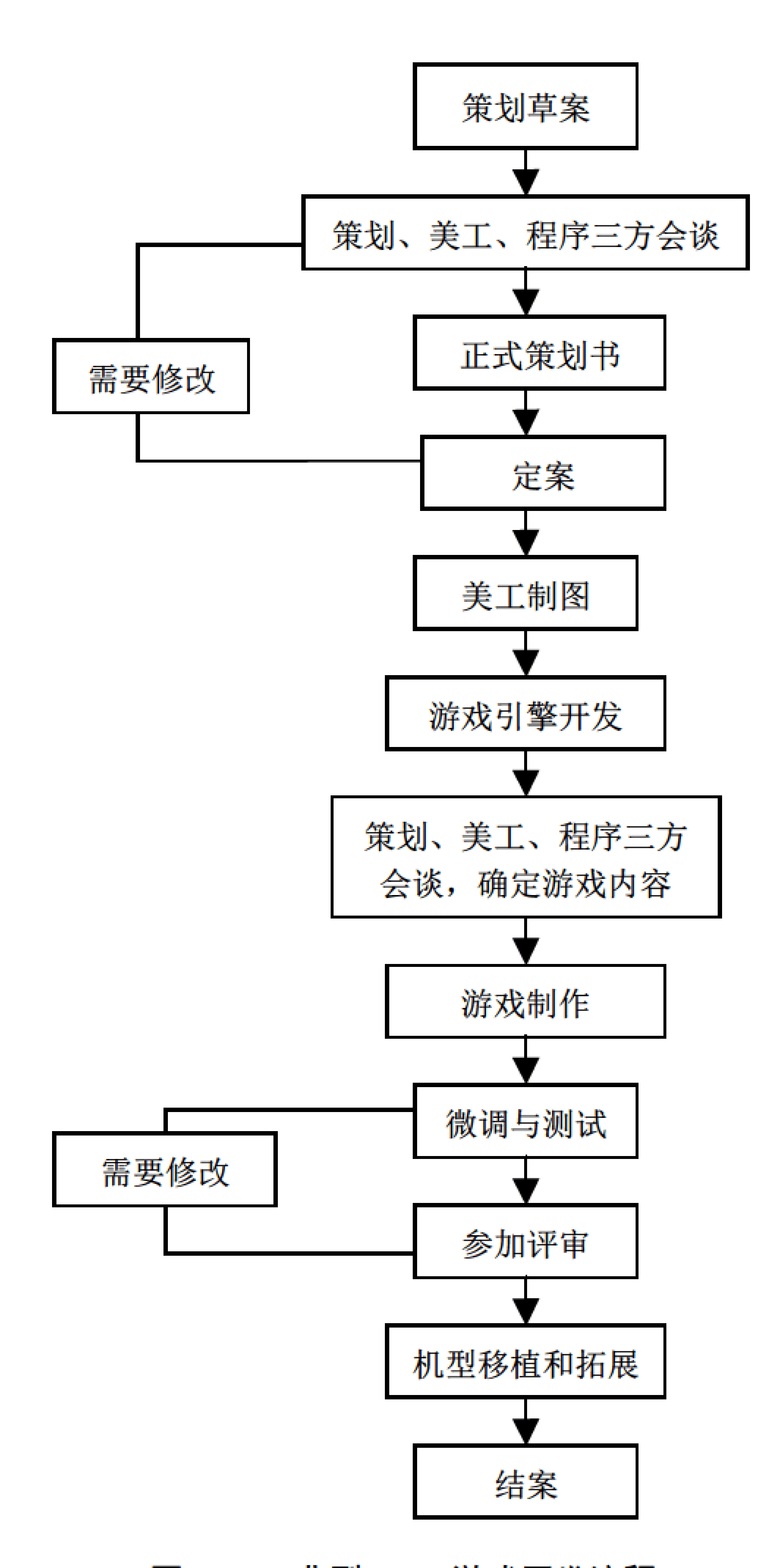


图 20-1　典型 Java 游戏开发流程

1. 立项

在制作游戏之前，策划首先要确定一点：到底想要制作一款什么样的游戏？而要制作一个游戏并不是闭门造车，一个策划说了就算数的简单事情。制作一款要游戏受到多方面的限制。

(1) 市场：即将做的游戏是不是具备市场潜力？在市场上推出以后会不会被大家所接受？是否能够取得良好的市场回报？

(2) 技术：即将做的游戏从程序和美术上是不是完全能够实现？如果不能实现，是不是能够有折中的办法？

(3) 规模：以现有的资源是否能很好地协调并完成即将要做的游戏？是否需要另外增加人员或设备？

(4) 周期：游戏的开发周期是否长短合适？能否在开发结束时正好赶上游戏的销售旺季？

(5) 产品：即将做的游戏在其同类产品中是否有新颖的设计？是否能有吸引玩家的地方？如果在游戏设计上达不到革新，是否能够在美术及程序方面加以补足？如果同类型的游戏市场

上已经很多，那么即将做的游戏的卖点在哪里？

以上各个问题都是要经过开发组全体成员反复进行讨论才能够确定下来的，大家一起集思广益，共同探讨一个可行的方案。如果对上述全部问题都能够有肯定的答案的话，那么这个项目基本是可行的。但是即便项目获得了通过，在进行过程中也可能会有种种不可预知的因素导致意外情况的发生，所以项目能够成立，只是游戏制作的开始。

在项目确立了以后，下一步要进行的就是进行游戏的大纲策划工作。

2. 大纲策划的进行

游戏大纲关系到游戏的整体面貌，当大纲策划案定稿以后，没有特别特殊的情况，是不允许进行更改的。程序和美术工作人员将按照策划所构思的游戏形式来架构整个游戏，因此，在制定策划案时一定要做到慎重和尽量考虑成熟。

3. 游戏的正式制作

当游戏大纲策划案完成并讨论通过后，游戏就由三方面同时开始进行制作了。在这一阶段，策划的主要任务是在大纲的基础上对游戏的所有细节进行完善，将游戏大纲逐步填充为完整的游戏策划案。根据不同的游戏种类，所要进行细化的部分也不尽相同。

在正式制作的过程中，策划、程序、美工人员进行及时和经常性的交流，了解工作进展以及是否有难以克服的困难，并且根据现实情况有目的的变更工作计划或设计思想。三方面的配合在游戏正式制作过程中是最重要的。

4. 配音、配乐

在程序和美工进行的差不多要结束的时候，就要进行配音和配乐的工作了。虽然音乐和音效是游戏的重要组成部分，能够起到很好的烘托游戏气氛的作用，但是限于 J2ME 游戏的开发成本和设置的处理能力，这部分已经被弱化到可有可无的地步了。但仍应选择跟游戏风格能很好配合的音乐当做游戏背景音乐，这个工作交给策划比较合适。

5. 检测、调试

游戏刚制作完成，肯定在程序上会有很多的错误，严重情况下会导致游戏完全没有办法进行下去。同样，策划的设计也会有不完善的地方，主要在游戏的参数部分。参数部分的不合理，会导致影响游戏的可玩性。此时测试人员需检测程序上的漏洞和通过试玩，调整游戏的各个部分参数使之基本平衡。

20.3 设计游戏框架

书归正传，现在开始讲解魔塔游戏的具体设计。因为所有游戏是基于框架的，所以设计一个合适的框架尤为重要。为了正确设计框架，我们先看市面上魔塔游戏的界面，如图 20-2 所示。

由游戏界面可知，游戏中包含了地图、角色、屏幕界面、道具等元素，上述元素构成了一个视图，例如屏幕视图、道具视图、角色视图等。

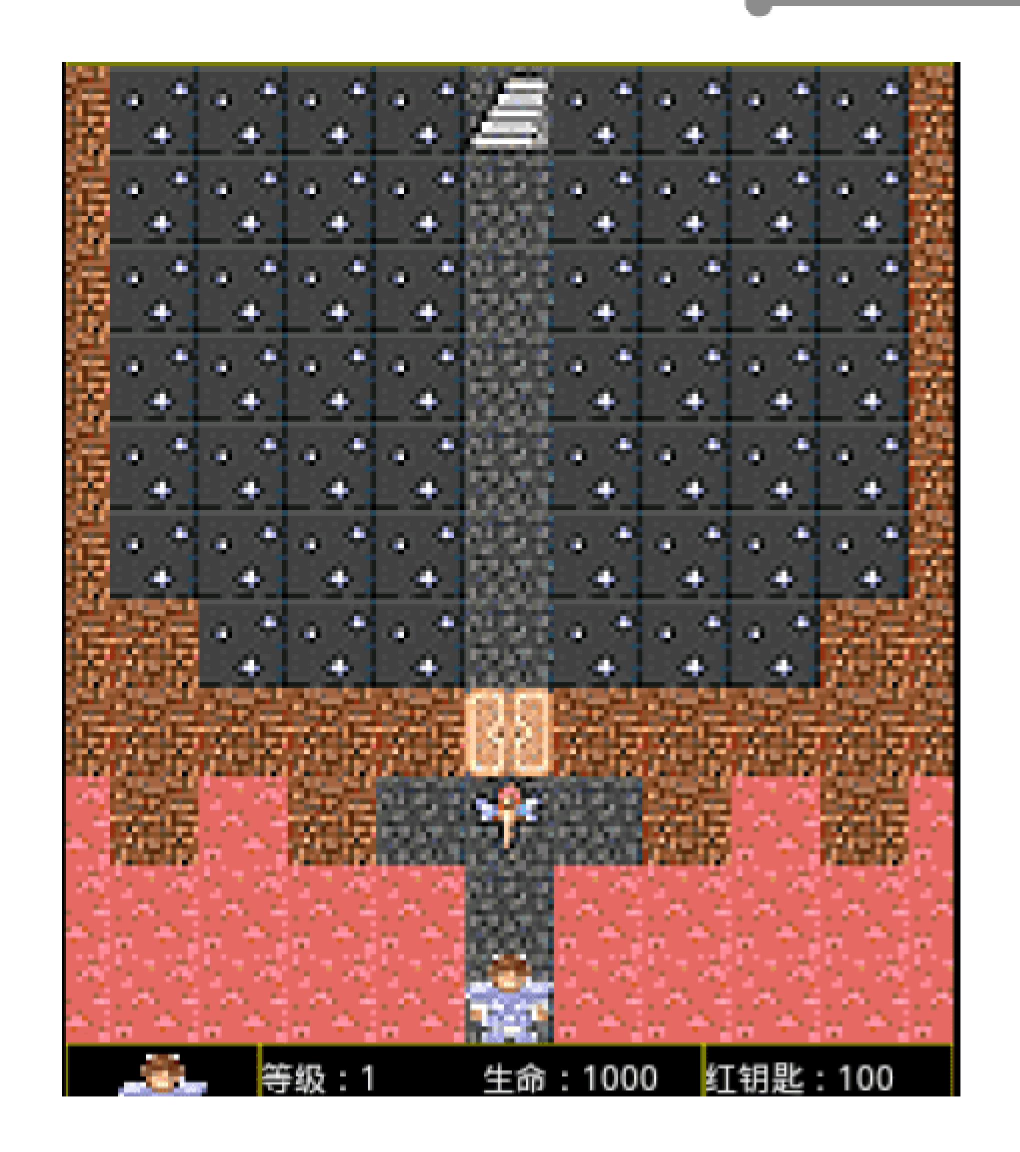


图 20-2　游戏界面

20.3.1　界面视图

在 Android 中，视图是通过继承 View 类实现的，在 View 类中包含了各种绘制图形的方法和事件处理，例如 onKeyDown、onKeyUp 等。在构造此视图类时还可以加入自己的一些抽象方法，例如资源回收和刷新等，有了这些内容，构建一个游戏界面类将变得轻而易举。

界面类 GameView 类的具体代码如下：

```
package com.example11;
import android.content.Context;
import android.graphics.Canvas;
import android.view.View;
public abstract class GameView extends View
{
    public GameView(Context context)
    {
        super(context);
    }
    /**
     * 绘图
     *
     * @param       N/A
     *
     * @return      null
     */
    protected abstract void onDraw(Canvas canvas);
    /**
     * 按键按下
     *
     * @param       N/A
     *
     * @return      null
```

```
     */
    public abstract boolean onKeyDown(int keyCode);
    /**
     * 按键弹起
     *
     * @param       N/A
     *
     * @return      null
     */
    public abstract boolean onKeyUp(int keyCode);
    /**
     * 回收资源
     *
     */
    protected abstract void reCycle();

    /**
     * 刷新
     *
     */
    protected abstract void refurbish();
}
```

20.3.2 屏幕显示

前面的视图类用于显示游戏界面，我们还需要控制当前的屏幕显示哪一个界面，并且能够对界面进行一些逻辑上的处理，这就需要建立一整个游戏的 MainGame 类，在此类中需要根据不同的游戏状态来设置屏幕需要显示的视图。在此编写 MainGame 类。具体代码如下：

```
package com.example11;
import android.app.Activity;
import android.content.Context;
public class MainGame
{
    private static GameView m_GameView = null;      // 当前需要显示的对象
    private Context m_Context = null;
    private MagicTower m_MagicTower= null;
    private int m_status = -1;                      //游戏状态
    public  CMIDIPlayer mCMIDIPlayer;
    public byte mbMusic = 0;
    public MainGame(Context context)
    {
        m_Context = context;
        m_MagicTower = (MagicTower)context;
        m_status = -1;
        initGame();
    }

    //初始化游戏
    public void initGame()
    {
```

```
        controlView(yarin.GAME SPLASH);
        mCMIDIPlayer = new CMIDIPlayer(m_MagicTower);
    }
    //得到游戏状态
    public int getStatus()
    {
        return m_status;
    }
    //设置游戏状态
    public void setStatus(int status)
    {
        m status = status;
    }
    //得到主类对象
    public Activity getMagicTower()
    {
        return m_MagicTower;
    }
    //得到当前需要显示的对象
    public static GameView getMainView()
    {
       return m_GameView;
    }
    //控制显示什么界面
    public void controlView(int status)
    {
        if(m_status != status)
        {
            if(m_GameView != null)
            {
                m_GameView.reCycle();
                System.gc();
            }
        }
        freeGameView(m_GameView);
        switch (status)
        {
        case yarin.GAME SPLASH:
            m_GameView = new SplashScreen(m_Context,this);
            break;
        case yarin.GAME_MENU:
            m_GameView = new MainMenu(m_Context,this);
            break;
        case yarin.GAME_HELP:
            m_GameView = new HelpScreen(m_Context,this);
            break;
        case yarin.GAME_ABOUT:
            m_GameView = new AboutScreen(m_Context,this);
            break;
        case yarin.GAME RUN:
            m GameView = new GameScreen(m Context,m MagicTower,this,true);
            break;
```

```
            case yarin.GAME CONTINUE:
                m_GameView = new GameScreen(m_Context,m_MagicTower,this,false);
                break;
            }
            setStatus(status);
        }
        //释放界面对象
        public void freeGameView(GameView gameView)
        {
            if(gameView != null)
            {
                gameView = null;
                System.gc();
            }
        }
}
```

20.3.3 线程更新

当创建和控制视图显示以后，屏幕界面设计告一段落，但是还需要让游戏能够动起来，要实现界面的更新则需要线程来实现，这样才能从本质上实现实时更新。在此可以为游戏开启一个主线程，并通过 MainGame.getMainView()方法来获取当前显示的界面，并根据不同的界面来进行游戏更新。此功能在文件 ThreadCanvas.java 中实现。具体代码如下：

```
package com.example11;

import android.content.Context;
import android.graphics.Canvas;
import android.util.Log;
import android.view.View;
public class ThreadCanvas extends View implements Runnable
{
    private String m_Tag = "ThreadCanvas_Tag";
    public ThreadCanvas(Context context)
    {
        super(context);
    }
    /**
     * 绘图
     *
     * @param N/A
     *
     *
     * @return null
     */
    protected void onDraw(Canvas canvas)
    {
        if (MainGame.getMainView() != null)
        {
            MainGame.getMainView().onDraw(canvas);
        }
```

```
    else
    {
        Log.i(m_Tag, "null");
    }
}

/**
 * 绘图显示
 *
 */
public void start()
{
    Thread t = new Thread(this);
    t.start();
}
// 刷新界面
public void refurbish()
{
    if (MainGame.getMainView() != null)
    {
        MainGame.getMainView().refurbish();
    }
}

/**
 * 游戏循环
 *
 * @param N/A
 * @return null
 */
public void run()
{
    while (true)
    {
        try
        {
            Thread.sleep(yarin.GAME LOOP);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

        refurbish();        // 更新显示
        postInvalidate();   // 刷新屏幕
    }
}
// 按键处理(按键按下)
boolean onKeyDown(int keyCode)
{
    if (MainGame.getMainView() != null)
    {
```

```
            MainGame.getMainView().onKeyDown(keyCode);
        }
        else
        {
            Log.i(m_Tag, "null");
        }
        return true;
    }
    // 按键弹起
    boolean onKeyUp(int keyCode)
    {
        if (MainGame.getMainView() != null)
        {
            MainGame.getMainView().onKeyUp(keyCode);
        }
        else
        {
            Log.i(m_Tag, "null");
        }
        return true;
    }
}
```

20.3.4　具体显示

经过上述类的设计后，框架就设计完毕了吗？当然不是，还需要调用一个 Activity 来显示具体的界面。因为是在 ThreadCanvas 中控制界面的，所以只需用 setContentView 来显示一个 ThreadCanvas 对象即可。上述功能通过文件 MagicTower.java 实现。具体代码如下：

```
package com.example11;
import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.Window;
import android.view.WindowManager;
public class MagicTower extends Activity
{
    private ThreadCanvas   mThreadCanvas   = null;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setTheme(android.R.style.Theme_Black_NoTitleBar_Fullscreen);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);
        new MainGame(this);
        mThreadCanvas = new ThreadCanvas(this);
        setContentView(mThreadCanvas);
    }
    /**
     * 暂停
```

```
     *
     * @param
     *
     * @return null
     */
    protected void onPause()
    {
        super.onPause();
    }
    /**
     * 重绘
     *
     * @param N/A
     *
     * @return null
     */
    protected void onResume()
    {
        super.onResume();
        mThreadCanvas.requestFocus();
        mThreadCanvas.start();
    }
    /**
     * 按键按下
     *
     * @param N/A
     *
     * @return null
     */
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        mThreadCanvas.onKeyDown(keyCode);
        return false;
    }
    /**
     * 按键弹起
     *
     * @param N
     *           /A
     *
     * @return null
     */
    public boolean onKeyUp(int keyCode, KeyEvent event)
    {
        mThreadCanvas.onKeyUp(keyCode);
        return false;
    }
}
```

通过上述处理，一个基本的游戏框架宣告建设完毕。后面的所有视图类界面都继承自自定义的抽象类 GameView，也就是说在后续的开发中，只需直接继承上面的类界面即可，即继承 GameView 类，然后在 MainView 中更改游戏状态即可。

20.4 后面的视图

经过前面的游戏框架设计之后，宣告整个项目的根基打好了。接下来需要我们在根基的基础上增砖添瓦，直到构建一个完整的游戏“建筑”为止。此魔塔游戏后面的功能也都是基于视图的，下面将详细介绍其实现过程。

20.4.1 设计地图

地图在游戏项目中十分重要，游戏中的所有角色都需要在某个场景中生存。但是，我们不可能让美工制作一张巨大的地图供我们使用，因为这样的游戏将会很大，不利于在手机有限的空间和配置中使用。通常游戏中的地图是由多个小块构成的一个完整大图，所以我们完全可以使用一个二维数组来存储小块数据，然后通过程序将地图数据对应的小块映射到屏幕，从而组成一幅完整的地图。当前主流的做法是用 mappy 来生成地图，然后用脚本语言为 mappy 写一个保存格式的程序。一般情况下，地图分为 45° 角、仰视角和俯视角。

地图的实现思路很清晰，具体流程如下。

(1) 创建一个对象类，在此命名为 TiledLayer。具体代码如下：

```
public class TiledLayer extends Layer
```

然后新建对应的构造函数。具体代码如下：

```
    public TiledLayer(int columns, int rows, Bitmap image, int tileWidth,
            int tileHeight) {
        super(columns < 1 || tileWidth < 1 ? -1 : columns * tileWidth, rows < 1
                || tileHeight < 1 ? -1 : rows * tileHeight);
        if (((image.getWidth() % tileWidth) != 0)
                || ((image.getHeight() % tileHeight) != 0)) {
            throw new IllegalArgumentException();
        }
        this.columns = columns;
        this.rows = rows;

        cellMatrix = new int[rows][columns];

        int noOfFrames = (image.getWidth() / tileWidth)
                * (image.getHeight() / tileHeight);
        createStaticSet(image, noOfFrames + 1, tileWidth, tileHeight, true);
    }
```

其中，参数 columns 和 rows 分别表示地图的行数和列数，image 代表地图土块中的一块，tileWidth 和 tileHeight 分别表示土块的宽度和高度。

(2) 定义方法 setCell(int col, int row, int tileIndex)，用于设置地图使用的地图数据。此方法的具体实现代码如下：

```
    public void setCell(int col, int row, int tileIndex) {
        if (col < 0 || col >= this.columns || row < 0 || row >= this.rows) {
```

```
            throw new IndexOutOfBoundsException();
        }
        if (tileIndex > 0) {
            if (tileIndex >= numberOfTiles) {
                throw new IndexOutOfBoundsException();
            }
        } else if (tileIndex < 0) {
            // do animated tile index check
            if (anim_to_static == null || (-tileIndex) >= numOfAnimTiles) {
                throw new IndexOutOfBoundsException();
            }
        }
        cellMatrix[row][col] = tileIndex;
    }
```

(3)　通过方法 createAnimatedTile(int staticTileIndex)实现动态贴图，此方法在 J2ME 中十分常见。动态贴图可以通过调用这个方法来创建，该方法返回一个索引号，用于标记新创建的动态贴图。动态贴图的索引号总是负数，并且也是连续的，起始值为-1。一旦被创建，与之关联的静态贴图可以通过调用 setAnimatedTile(int, int)方法来改变。具体代码如下：

```
    public int createAnimatedTile(int staticTileIndex) {
        // checks static tile
        if (staticTileIndex < 0 || staticTileIndex >= numberOfTiles) {
            throw new IndexOutOfBoundsException();
        }
        if (anim_to_static == null) {
            anim_to_static = new int[4];
            numOfAnimTiles = 1;
        } else if (numOfAnimTiles == anim_to_static.length) {
            // grow anim_to_static table if needed
            int new_anim_tbl[] = new int[anim_to_static.length * 2];
            System.arraycopy(anim_to_static, 0, new_anim_tbl, 0,
                    anim_to_static.length);
            anim_to_static = new_anim_tbl;
        }
        anim_to_static[numOfAnimTiles] = staticTileIndex;
        numOfAnimTiles++;
        return (-(numOfAnimTiles - 1));
    }
```

(4)　定义函数 setAnimatedTile(int animatedTileIndex, int staticTileIndex)，动态图块的内容就是使用这个函数改变的，函数的第一个参数是动态图块的编号，第二个参数是 Tile 的编号。具体代码如下：

```
        public void setAnimatedTile(int animatedTileIndex, int staticTileIndex)
{
        if (staticTileIndex < 0 || staticTileIndex >= numberOfTiles) {
            throw new IndexOutOfBoundsException();
        }
                animatedTileIndex = -animatedTileIndex;
        if (anim_to_static == null || animatedTileIndex <= 0
                || animatedTileIndex >= numOfAnimTiles) {
```

```
            throw new IndexOutOfBoundsException();
        }

        anim_to_static[animatedTileIndex] = staticTileIndex;

    }
```

(5) 定义函数 getAnimatedTile(int animatedTileIndex)，其功能是得到序号为 animatedTileIndex 的动画分块实际的图像分块序号。具体代码如下：

```
    public int getAnimatedTile(int animatedTileIndex) {
        animatedTileIndex = -animatedTileIndex;
        if (anim_to_static == null || animatedTileIndex <= 0
                || animatedTileIndex >= numOfAnimTiles) {
            throw new IndexOutOfBoundsException();
        }
        return anim_to_static[animatedTileIndex];
    }
```

(6) 通过 fillCells()方法将图绘制在屏幕上。具体代码如下：

```
    public void fillCells(int col, int row, int numCols, int numRows,
            int tileIndex) {

        if (col < 0 || col >= this.columns || row < 0 || row >= this.rows
                || numCols < 0 || col + numCols > this.columns || numRows < 0
                || row + numRows > this.rows) {
            throw new IndexOutOfBoundsException();
        }

        if (tileIndex > 0) {
            if (tileIndex >= numberOfTiles) {
                throw new IndexOutOfBoundsException();
            }
        } else if (tileIndex < 0) {
            if (anim_to_static == null || (-tileIndex) >= numOfAnimTiles) {
                throw new IndexOutOfBoundsException();
            }
        }

        for (int rowCount = row; rowCount < row + numRows; rowCount++) {
           for (int columnCount = col; columnCount < col + numCols; columnCount++)
{
                cellMatrix[rowCount][columnCount] = tileIndex;
            }
        }
    }
```

注意： 地图的建立还用到了其他知识，这些知识都属于 MIDP 的范畴，这不是本书所要讲解的内容。在表 20-1 和表 20-2 中简单介绍 MIDP 中的常用方法，至于具体知识希望读者利用课外时间学习。

表 20-1　Layer 类所定义的方法

方法名称	方法原型与作用
getHeight	public int getHeight( ) 得到图层的高度
getWidth	public int getWidth( ) 得到图层的宽度
move	public void move(int dx, int dy) 移动图层，参数 dx 和 dy 分别表明在 X 轴和 Y 轴上移动的距离。如果 dx 距离小于零，则表示向左或向上移动；如果 dy 距离大于零，则表示向右或向下移动
getX	public int getX( ) 得到图层的起始点 X 坐标
getY	public int getY( ) 得到图层的起始点 Y 坐标
setPosition	public void setPosition(int x, int y) 将图层移动到参数(x, y)指定的坐标处
setVisible	public void setVisible(boolean visible) 设置图层是否可见
isVisible	public boolean isVisible( ) 检测图层是否可见
paint	public abstract void paint(Graphics g) 绘制图层，只有对于可见的图层才会被实际绘制。一般来说，不需要直接调用图层对象的 paint 方法，而是由 LayerManager 对象调用这个方法

表 20-2　TiledLayer 类所包含的方法

方法名称	方法原型与作用
TiledLayer	public TiledLayer(int columns, int rows, Image image, int tileWidth, int tileHeight) 构造方法，参数 columns 和 rows 分别表明图层中的单元格列数和行数，参数 image 为包含所有分块的图像，参数 tileWidth 和 tileHeight 分别表明每个分块的宽度和高度。图层的面积是单元格的数量乘以每个分块图像的面积。图层实际的宽度为 columns × (tileWidth × tileHeight) ，高度为 rows × (tileWidth × tileHeight)
setCell	public void setCell(int col, int row, int tileIndex) 将图层中指定位置(col，row)单元格设置为序号为参数 tileIndex 指定的分块图像。tileIndex 必须为 0 或者是正数。如果为 0 表示该单元格不填充
fillCells	public void fillCells(int col, int row, int numCols, int numRows, int tileIndex) 将图层中指定的单元格区域，位置为(col, row)、面积为(numCols, numRows) 用序号为 tileIndex 分块图像填充。fillCells 与 setCell 方法不同之处在于，能够一次设置多个相同的单元格
getCellWidth	public int getCellWidth( ) 得到单元格宽度，也就是构造对象时 tileWidth 的值
getCellHeight	public int getCellHeight( ) 得到单元格高度，也就是构造对象时 tileHeight 的值

续表

方法名称	方法原型与作用
getColumns	public int getColumns( ) 得到图层中单元格的列数
getRows	public int getRows( ) 得到图层中单元格的行数
setStaticTileSet	public void setStaticTileSet(Image image, int tileWidth, int tileHeight) 重新创建分块图像。在执行此操作后对象所包含的原有分块会被删除，而且图层的面积会根据新的分块大小重新计算
CreateAnimatedTile	public int createAnimatedTile(int staticTileIndex) 创建一个动画分块，初始化时把序号为 staticTileIndex 的分块图像作为这个动画分块的图像。返回值为创建的动画单元格序号，第一个创建动画单元格为 −1，第二个为 −2
setAnimatedTile	public void setAnimatedTile(int animatedTileIndex, int staticTileIndex) 将序号为 animatedTileIndex 的动画分块的图像用序号为 staticTileIndex 的分块图像代替
getAnimatedTile	public int getAnimatedTile(int animatedTileIndex) 得到序号为 animatedTileIndex 的动画分块实际的图像分块序号

20.4.2　设计主角

游戏中会有一个主角，在魔塔游戏中我们称之为“精灵”，并且这个角色还有很多的动作，例如向四个方向的移动，在移动的时候就是一个动画。动画本身是将图片一帧一帧地连接起来、循环播放每一帧所形成的。在一些大型游戏中，可以使用精灵编辑器去编写精灵，将精灵拆解为很多部分，然后再组合起来，这样可以节省大量的空间。在此我决定使用 Sprite 类实现魔塔中的主角，此类是一个用于显示图像的类。下面开始讲解主角的具体实现过程。

(1)　构建 Sprite 对象，先定义 Sprite 类。具体代码如下：

```
public class Sprite extends Layer
```

在 Sprite 类中提供了 3 个构造方法来构建完整的 Sprite 类，其中方法 Sprite(Bitmap image)的主要代码如下：

```
    public Sprite(Bitmap image) {
        super(image.getWidth(), image.getHeight());
        initializeFrames(image, image.getWidth(), image.getHeight(), false);
        initCollisionRectBounds();
        this.setTransformImpl(TRANS_NONE);

    }
```

方法 Sprite(Bitmap image, int frameWidth, int frameHeight)的具体实现代码如下：

```
    public Sprite(Bitmap image, int frameWidth, int frameHeight) {
        super(frameWidth, frameHeight);
```

```
        if ((frameWidth < 1 || frameHeight < 1)
                || ((image.getWidth() % frameWidth) != 0)
                || ((image.getHeight() % frameHeight) != 0)) {
            throw new IllegalArgumentException();
        }
        initializeFrames(image, frameWidth, frameHeight, false);
        initCollisionRectBounds();
        this.setTransformImpl(TRANS_NONE);
    }
```

方法 Sprite(Sprite s)的具体实现代码如下：

```
    public Sprite(Sprite s) {
        super(s != null ? s.getWidth() : 0, s != null ? s.getHeight() : 0);
        if (s == null) {
            throw new NullPointerException();
        }
        this.sourceImage = s.sourceImage;
        this.numberFrames = s.numberFrames;
        this.frameCoordsX = new int[this.numberFrames];
        this.frameCoordsY = new int[this.numberFrames];
        System.arraycopy(s.frameCoordsX, 0, this.frameCoordsX, 0, s
                .getRawFrameCount());
        System.arraycopy(s.frameCoordsY, 0, this.frameCoordsY, 0, s
                .getRawFrameCount());
        this.x = s.getX();
        this.y = s.getY();
        this.dRefX = s.dRefX;
        this.dRefY = s.dRefY;
        this.collisionRectX = s.collisionRectX;
        this.collisionRectY = s.collisionRectY;
        this.collisionRectWidth = s.collisionRectWidth;
        this.collisionRectHeight = s.collisionRectHeight;
        this.srcFrameWidth = s.srcFrameWidth;
        this.srcFrameHeight = s.srcFrameHeight;
        setTransformImpl(s.t_currentTransformation);
        this.setVisible(s.isVisible());
        this.frameSequence = new int[s.getFrameSequenceLength()];
        this.setFrameSequence(s.frameSequence);
        this.setFrame(s.getFrame());

        this.setRefPixelPosition(s.getRefPixelX(), s.getRefPixelY());
    }
```

在上述 3 个构造函数中，参数 image 为精灵的图片，参数 frameWidth 和 frameHeight 分别设置了精灵图片每一帧的宽度和高度，参数 s 表示通过一个精灵来创建另一个精灵。构造 Sprite 类的时候需要指定精灵的高度和宽度(像素值)。图像的高度和宽度必须分别是精灵的高度和宽度的整数倍。换句话说，要能正好让电脑把图像按照精灵的大小划分成几个类，通过上面的例子也看到了，这些帧是横着排还是竖着排，抑或横竖都有，排成一个方阵，都无所谓。接着就可以指定帧数了，左上方是编号 0，然后从左到右、从上到下依次排列。可以使用 setFrame(int sequenceIndex)来选择哪一帧被显示，只要把它的编号作为参数传递即可。

(2) 设置 Sprite 属性。

其实，TiledLayer 类可以自动根据精灵的位置来判断地图绘制的位置，因此可以使用一个方法来设置精灵的位置。在此定义为 setRefPixelPosition(int x, int y)方法。具体代码如下：

```
public void setRefPixelPosition(int x, int y) {
    // update this.x and this.y
    this.x = x getTransformedPtX(dRefX, dRefY, this.t currentTransformation);
    this.y = y - getTransformedPtY(dRefX, dRefY, this.t currentTransformation);
}
```

其中，参数 x 和 y 是精灵的位置。

(3) 实现碰撞检测处理。

碰撞即相遇，当精灵和外物相碰撞时就需要对应的处理。我的精灵类提供了以下 3 个碰撞检测函数：

- public final boolean collidesWith(TiledLayer t, boolean pixelLevel);
- public final boolean collidesWith(Sprite s, boolean pixelLevel);
- public final boolean collidesWith(Bitmap image, int x, int y,boolean pixelLevel)。

其中函数 collidesWith(TiledLayer t, boolean pixelLevel)实现精灵和 TiledLayer 的碰撞。具体代码如下：

```
public final boolean collidesWith(TiledLayer t, boolean pixelLevel) {
    if (!(t.visible && this.visible)) {
        return false;
    }
    int tLx1 = t.x;
    int tLy1 = t.y;
    int tLx2 = tLx1 + t.width;
    int tLy2 = tLy1 + t.height;
    int tW = t.getCellWidth();
    int tH = t.getCellHeight();
    int sx1 = this.x + this.t_collisionRectX;
    int sy1 = this.y + this.t_collisionRectY;
    int sx2 = sx1 + this.t_collisionRectWidth;
    int sy2 = sy1 + this.t collisionRectHeight;
    int tNumCols = t.getColumns();
    int tNumRows = t.getRows();
    int startCol; // = 0;
    int endCol; // = 0;
    int startRow; // = 0;
    int endRow; // = 0;
    if (!intersectRect(tLx1, tLy1, tLx2, tLy2, sx1, sy1, sx2, sy2)) {
        return false;
    }
    startCol = (sx1 <= tLx1) ? 0 : (sx1 - tLx1) / tW;
    startRow = (sy1 <= tLy1) ? 0 : (sy1 - tLy1) / tH;
    endCol = (sx2 < tLx2) ? ((sx2 - 1 - tLx1) / tW) : tNumCols - 1;
    endRow = (sy2 < tLy2) ? ((sy2 - 1 - tLy1) / tH) : tNumRows - 1;
    if (!pixelLevel) {
            for (int row = startRow; row <= endRow; row++) {
        for (int col = startCol; col <= endCol; col++) {
```

```
                if (t.getCell(col, row) != 0) {
                    return true;
                }
            }
        }
        return false;
    } else {
        if (this.t_collisionRectX < 0) {
            sx1 = this.x;
        }
        if (this.t_collisionRectY < 0) {
            sy1 = this.y;
        }
        if ((this.t_collisionRectX + this.t_collisionRectWidth) > this.width) {
            sx2 = this.x + this.width;
        }
        if ((this.t_collisionRectY + this.t_collisionRectHeight) > this.height) {
            sy2 = this.y + this.height;
        }
        if (!intersectRect(tLx1, tLy1, tLx2, tLy2, sx1, sy1, sx2, sy2)) {
            return (false);
        }
        startCol = (sx1 <= tLx1) ? 0 : (sx1 - tLx1) / tW;
        startRow = (sy1 <= tLy1) ? 0 : (sy1 - tLy1) / tH;
        endCol = (sx2 < tLx2) ? ((sx2 - 1 - tLx1) / tW) : tNumCols - 1;
        endRow = (sy2 < tLy2) ? ((sy2 - 1 - tLy1) / tH) : tNumRows - 1;
        int cellTop = startRow * tH + tLy1;
        int cellBottom = cellTop + tH;
        int tileIndex; // = 0;
        for (int row = startRow; row <= endRow; row++, cellTop += tH,
            cellBottom += tH) {
            int cellLeft = startCol * tW + tLx1;
            int cellRight = cellLeft + tW;
            for (int col = startCol; col <= endCol; col++, cellLeft += tW,
             cellRight += tW) {
                tileIndex = t.getCell(col, row);
                if (tileIndex != 0) {
                    int intersectLeft = (sx1 < cellLeft) ? cellLeft : sx1;
                    int intersectTop = (sy1 < cellTop) ? cellTop : sy1;
                    int intersectRight = (sx2 < cellRight) ? sx2
                            : cellRight;
                    int intersectBottom = (sy2 < cellBottom) ? sy2
                            : cellBottom;
                    if (intersectLeft > intersectRight) {
                        int temp = intersectRight;
                        intersectRight = intersectLeft;
                        intersectLeft = temp;
                    }
                    if (intersectTop > intersectBottom) {
                        int temp = intersectBottom;
                        intersectBottom = intersectTop;
                        intersectTop = temp;
```

```
                        }
                        int intersectWidth = intersectRight - intersectLeft;
                        int intersectHeight = intersectBottom - intersectTop;
                        int image1XOffset = getImageTopLeftX(intersectLeft,
                                intersectTop, intersectRight, intersectBottom);
                        int image1YOffset = getImageTopLeftY(intersectLeft,
                                intersectTop, intersectRight, intersectBottom);
                        int image2XOffset = t.tileSetX[tileIndex]
                                + (intersectLeft - cellLeft);
                        int image2YOffset = t.tileSetY[tileIndex]
                                + (intersectTop - cellTop);
                        if (doPixelCollision(image1XOffset, image1YOffset,
                                image2XOffset, image2YOffset, this.sourceImage,
                                this.t_currentTransformation, t.sourceImage,
                                TRANS_NONE, intersectWidth, intersectHeight)) {
                            return true;
                        }
                    }
                }
            }
            return false;
        }
    }
```

函数 collidesWith(Sprite s, boolean pixelLevel)实现精灵和精灵的碰撞。具体代码如下：

```
    public final boolean collidesWith(Sprite s, boolean pixelLevel) {
        if (!(s.visible && this.visible)) {
            return false;
        }
        int otherLeft = s.x + s.t_collisionRectX;
        int otherTop = s.y + s.t_collisionRectY;
        int otherRight = otherLeft + s.t_collisionRectWidth;
        int otherBottom = otherTop + s.t collisionRectHeight;

        int left = this.x + this.t_collisionRectX;
        int top = this.y + this.t_collisionRectY;
        int right = left + this.t_collisionRectWidth;
        int bottom = top + this.t_collisionRectHeight;
        if (intersectRect(otherLeft, otherTop, otherRight, otherBottom, left,
                top, right, bottom)) {
            if (pixelLevel) {
                if (this.t_collisionRectX < 0) {
                    left = this.x;
                }
                if (this.t_collisionRectY < 0) {
                    top = this.y;
                }
                if ((this.t_collisionRectX + this.t_collisionRectWidth) >
                this.width) {
                    right = this.x + this.width;
                }
```

```
                if ((this.t collisionRectY + this.t collisionRectHeight) >
                this.height) {
                    bottom = this.y + this.height;
                }
                if (s.t_collisionRectX < 0) {
                    otherLeft = s.x;
                }
                if (s.t_collisionRectY < 0) {
                    otherTop = s.y;
                }
                if ((s.t_collisionRectX + s.t_collisionRectWidth) > s.width) {
                    otherRight = s.x + s.width;
                }
                if ((s.t_collisionRectY + s.t_collisionRectHeight) > s.height) {
                    otherBottom = s.y + s.height;
                }
                if (!intersectRect(otherLeft, otherTop, otherRight,
                        otherBottom, left, top, right, bottom)) {
                    return false;
                }
                int intersectLeft = (left < otherLeft) ? otherLeft : left;
                int intersectTop = (top < otherTop) ? otherTop : top;
                int intersectRight = (right < otherRight) ? right : otherRight;
                int intersectBottom = (bottom < otherBottom) ? bottom
                        : otherBottom;

                int intersectWidth = Math.abs(intersectRight - intersectLeft);
                int intersectHeight = Math.abs(intersectBottom - intersectTop);
                int thisImageXOffset = getImageTopLeftX(intersectLeft,
                        intersectTop, intersectRight, intersectBottom);
                int thisImageYOffset = getImageTopLeftY(intersectLeft,
                        intersectTop, intersectRight, intersectBottom);
                int otherImageXOffset = s.getImageTopLeftX(intersectLeft,
                        intersectTop, intersectRight, intersectBottom);
                int otherImageYOffset = s.getImageTopLeftY(intersectLeft,
                        intersectTop, intersectRight, intersectBottom);
                return doPixelCollision(thisImageXOffset, thisImageYOffset,
                        otherImageXOffset, otherImageYOffset, this.sourceImage,
                        this.t_currentTransformation, s.sourceImage,
                        s.t_currentTransformation, intersectWidth,
                        intersectHeight);

            } else {
            return true;
            }
        }
        return false;
    }
```

函数 collidesWith(Bitmap image, int x, int y,boolean pixelLevel)用于实现精灵和图片的碰撞。具体代码如下：

```
    public final boolean collidesWith(Bitmap image, int x, int y,
            boolean pixelLevel) {
        if (!(this.visible)) {
            return false;
        }
        int otherLeft = x;
        int otherTop = y;
        int otherRight = x + image.getWidth();
        int otherBottom = y + image.getHeight();

        int left = this.x + this.t_collisionRectX;
        int top = this.y + this.t_collisionRectY;
        int right = left + this.t_collisionRectWidth;
        int bottom = top + this.t_collisionRectHeight;
        if (intersectRect(otherLeft, otherTop, otherRight, otherBottom, left,
                top, right, bottom)) {
            if (pixelLevel) {
                if (this.t_collisionRectX < 0) {
                    left = this.x;
                }
                if (this.t_collisionRectY < 0) {
                    top = this.y;
                }
                if ((this.t_collisionRectX + this.t_collisionRectWidth) > this.width) {
                    right = this.x + this.width;
                }
                if ((this.t_collisionRectY + this.t_collisionRectHeight) > this.height) {
                    bottom = this.y + this.height;
                }
                if (!intersectRect(otherLeft, otherTop, otherRight,
                        otherBottom, left, top, right, bottom)) {
                    return false;
                }
                int intersectLeft = (left < otherLeft) ? otherLeft : left;
                int intersectTop = (top < otherTop) ? otherTop : top;
                int intersectRight = (right < otherRight) ? right : otherRight;
                int intersectBottom = (bottom < otherBottom) ? bottom
                        : otherBottom;
                int intersectWidth = Math.abs(intersectRight - intersectLeft);
                int intersectHeight = Math.abs(intersectBottom - intersectTop);
                int thisImageXOffset = getImageTopLeftX(intersectLeft,
                        intersectTop, intersectRight, intersectBottom);
                int thisImageYOffset = getImageTopLeftY(intersectLeft,
                        intersectTop, intersectRight, intersectBottom);
                int otherImageXOffset = intersectLeft - x;
                int otherImageYOffset = intersectTop - y;
                return doPixelCollision(thisImageXOffset, thisImageYOffset,
                        otherImageXOffset, otherImageYOffset, this.sourceImage,
                        this.t_currentTransformation, image, Sprite.TRANS_NONE,
                        intersectWidth, intersectHeight);
            } else {
                return true;
```

```
            }
        }
        return false;
    }
```

在上述三个函数中，参数 pixelLevel 表示使用像素检测还是矩形检测。矩形检测只需要将精灵对应成相应的矩形范围来进行检查，这种检测速度很快，但不是很准确，对于碰撞要求不高的游戏可以使用它。同时还可以将一个 Sprite 分解成很多矩形来使用矩形检测以提高准确性，而像素检测则比较准确，但是速度必然会减慢。

(4)　实现简单的主角对话。

我设计的《魔塔》主角可以和 NPC 对话来获得一些信息，然后进行游戏。我准备通过一个浮动的对话框来显示对话内容，这个对话框只是一个矩形框，然后在右边绘制出对应的 NPC 的头像即可。这里的对话内容可以通过前面介绍的 TextUtil 类来实现自动换行。

(5)　实现精灵旋转和镜像。

在游戏开发时，一般都需要使用旋转和镜像。例如，做一个飞机游戏时，飞机会有几个方向的图片，这样会增加游戏开发出来的包的大小，所以可以制作一个方向的图片，然后通过精灵的旋转方法来将图片在各个方向进行旋转。这里我们只实现了 90° 的倍数旋转，分别是我们在 Sprite 类中定义的常量：TRANS_NONE、TRANS_ROT90、TRANS_ROT180、TRANS_ROT270、TRANS_MIRROR、TRANS_MIRROR_ROT90、TRANS_MIRROR_ROT180、TRANS_MIRROR_ROT270。我可以通过 setTransform()方法来传输这些常量，设置精灵的旋转和镜像。当然，可以查看该方法的具体实现来更深入地理解精灵的旋转和镜像。

注意：上面两个功能比较简单，所以不再列出详细代码，请读者参考本书光盘中的源代码。

(6)　实现战斗界面。

既然是闯关游戏，则战斗界面不可避免。当主角和怪物发生碰撞时就会发生战斗，这时需要一个界面来显示战斗效果。我设计的战斗界面很简单，就是分别显示玩家和怪物的头像以及属性，包括生命、攻击、防御。此功能是由文件 FightScreen.java 实现的，其中绘制战斗界面功能的实现代码如下：

```
    protected void onDraw(Canvas canvas)
    {
        mcanvas = canvas;
        int tx, ty, tw, th;
        tw = yarin.SCREENW;
        th = yarin.MessageBoxH;
        tx = 0;
        ty = (yarin.SCREENH - yarin.MessageBoxH) / 2;
        showMessage();
        if (!isFighting)
        {
            tu.DrawText(mcanvas);
        }
        else
        {
            yarin.drawImage(canvas, orgeImage, 0, ty + (th - GameMap.TILE_WIDTH)
            / 2, GameMap.TILE_WIDTH, GameMap.TILE_WIDTH, orgeSrcX, orgeSrcY);
```

```
            yarin.drawImage(canvas, heroImage, (tw - GameMap.TILE WIDTH), ty +
            (th - GameMap.TILE_WIDTH) / 2, GameMap.TILE_WIDTH, GameMap.
            TILE_WIDTH, 0, 0);
            paint.setColor(Color.WHITE);
            // 怪物
            {
                tx = 40;
                ty = (yarin.SCREENH - yarin.MessageBoxH) / 2 + 5;
                yarin.drawString(canvas, "生命:" + orgeHp, tx, ty, paint);
                yarin.drawString(canvas, "攻击:" + orgeAttack, tx, ty +
                yarin.TextSize, paint);
                yarin.drawString(canvas, "防御:" + orgeDefend, tx, ty + 2 *
                yarin.TextSize, paint);
            }
            // 英雄
            {
                String string = "";
                ty = (yarin.SCREENH - yarin.MessageBoxH) / 2 + 5;
                string = hero.getHp() + ":生命";
                yarin.drawString(canvas, string, (tw - 40 - paint.measureText(string)),
                ty, paint);
                string = hero.getAttack() + ":攻击";
                yarin.drawString(canvas, string, (tw - 40 - paint.measureText(string)),
                ty + yarin.TextSize, paint);
                string = hero.getDefend() + ":防御";
                yarin.drawString(canvas, string, (tw - 40 - paint.measureText(string)),
                ty + 2 * yarin.TextSize, paint);
            }
        }

        tick();
    }
    public void showMessage()
    {
        int x = 0;
        int y = (yarin.SCREENH - yarin.MessageBoxH) / 2;
        int w = yarin.SCREENW;
        int h = yarin.MessageBoxH;
        Paint ptmPaint = new Paint();
        ptmPaint.setARGB(255, 0, 0, 0);

        yarin.fillRect(mcanvas, x, y, w, h, ptmPaint);
        ptmPaint = null;
    }
```

每次战斗过后都会得到经验值，经验值增加功能的实现代码如下：

```
    private void tick()
    {
        if (orgeHp <= 0)
        {
            isFighting = false;
```

```
            tu.InitText("得到" + orgeMoney + "个金币  " + "经验值增加" +
            orgeExperience, 0, (yarin.SCREENH - yarin.MessageBoxH) / 2,
            yarin.SCREENW, yarin.MessageBoxH,
                0x0, 0xff0000, 255, yarin.TextSize);
        }
        else if (heroFirst == true)
        {
            orgeHp -= orgeDamagePerBout;
            if (orgeHp <= 0)
            {
                orgeHp = 0;
            }
        }
        else
        {
            hero.cutHp(heroDamagePerBout);
        }
        heroFirst = !heroFirst;
    }
```

接下来还需要将 Sprite 显示在屏幕上，为此我在文件 Sprite.java 中编写了函数 paint(Canvas canvas)。具体代码如下：

```
    public final void paint(Canvas canvas) {
                if (canvas == null) {
            throw new NullPointerException();
        }

        if (visible) {
    drawImage(canvas,this.x,this.y,sourceImage,frameCoordsX[frameSequence
                 [sequenceIndex]],frameCoordsY[frameSequence[sequenceIndex]],
                     srcFrameWidth,
                     srcFrameHeight);
        }
    }
```

在上述代码中，通过 Visible 来检测是否需要显示当前精灵，如果需要显示我们才绘制，绘制时只需要指定要绘制精灵的位置、精灵的图片、当前帧在图片上的偏移量、帧的宽度和高度。

注意： Sprite 的编号是从 0 开始的，但是 TiledLayer 却是从 1 开始的。在 TiledLayer 中，序号 0 表示一个空白的元素(比如在某个位置你什么都不想画，那就把它设置成 0)。Sprite 只由一个单元组成，所以如果你想让它不显示这个单元，简单地设置成 setVisible(false)就可以了，因而 Sprite 不需要一个特殊的编号来表示空白的单元。

20.4.3　游戏音效

音效在游戏开发中起了很重要的作用，在开发游戏时，人们常常忽视游戏的音效。开发者往往把主要精力花费在游戏的图像和动画等方面，而忽视了背景音乐和声音效果。这种做法是不可取的，因为好的游戏音效和音乐可以使玩家融入游戏世界、产生共鸣。音效的作用还不仅限于此。如果没有高超的游戏音效的映衬，再好的图像技巧也无法使游戏的表现摆脱平庸，对

玩家也没有足够的吸引力。首先我们将游戏中的音效分为如下几类：背景音乐、剧情音乐、音效(动作的音效、使用道具音效、辅助音效)等。背景音乐一般需要一直播放，而剧情音乐则只需要在剧情需要的时候播放，音效则是很短小的一段，比如挥刀的声音、怪物叫声等。我准备为此游戏添加两个背景音乐：一个是菜单背景音乐；一个是游戏中的背景音乐。具体操作流程如下。

(1) 准备两个符合游戏剧情的背景音乐放到 res\raw 文件夹下面。

(2) 创建一个 CMIDIPlayer 类，控制音乐播放。

因为在 Android 中是通过 MediaPlayer 来播放音乐的，所以在 CMIDIPlayer 类中需要构建一个 MediaPlayer 对象，通过 MediaPlayer.create 来装载音乐文件，实现文件是 CMIDIPlayer.java。主要实现代码如下：

```
public class CMIDIPlayer
{
    public MediaPlayer playerMusic;
    public MagicTower magicTower = null;
    public CMIDIPlayer(MagicTower magicTower)
    {
        this.magicTower = magicTower;
    }
    // 播放音乐
    public void PlayMusic(int ID)
    {
        FreeMusic();
        switch (ID)
        {
            case 1:
                //装载音乐
                playerMusic = MediaPlayer.create(magicTower, R.raw.menu);
                //设置循环
                playerMusic.setLooping(true);
                try
                {
                    //准备
                    playerMusic.prepare();
                }
                catch (IllegalStateException e)
                {
                    e.printStackTrace();
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
                //开始
                playerMusic.start();
                break;
            case 2:
                playerMusic = MediaPlayer.create(magicTower, R.raw.run);
                playerMusic.setLooping(true);
```

```
                try
                {
                    playerMusic.prepare();
                }
                catch (IllegalStateException e)
                {
                    e.printStackTrace();
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
                playerMusic.start();
                break;
        }
    }
    // 退出释放资源
    public void FreeMusic()
    {
        if (playerMusic != null)
        {
            playerMusic.stop();
            playerMusic.release();
        }
    }
    // 停止播放
    public void StopMusic()
    {
        if (playerMusic != null)
        {
            playerMusic.stop();
        }
    }
}
```

在上述代码中，通过 PlayMusic 加上 ID 来确定播放什么音乐，通过 StopMusic 来停止正在播放的音乐，当程序退出时调用 FreeMusic()方法来释放播放音乐产生的资源。

(3)　创建“是否开启音效”界面。

在游戏中不可能强制玩家接受要播放的音乐，所以设计一个界面来供玩家选择是否开启音乐很有必要。播放音乐代码如下：

```
mCMIDIPlayer.PlayMusic(1);
```

mCMIDIPlayer 是我们构建的一个 CMIDIPlayer 类的对象。在进入游戏时，又需要播放另一首背景音乐。和上面的代码一样，只需要通过参数的 ID 来设置要播放的音乐。

(4)　释放资源。

当退出游戏时，需要释放资源，这时调用 CMIDPlayer 类的 FreeMusic 方法来释放资源，代码如下：

```
mCMIDIPlayer.FreeMusic();
```

在大多数游戏中，控制音效的界面也不只是这一个，可以在主菜单界面里设置音效，同样还可以在游戏中通过一个弹出菜单等来控制音效。但是实现方式都相同，大家可以自己将其完善，尽可能地方便用户随时控制音乐开关。

至此，我的足球游戏宣告一个段落，当然整个游戏的实现还不仅如此，还有很多重要的功能没有讲解，例如，游戏存档等。因为篇幅有限，所以在此不再进行详细讲解，希望读者仔细品味本书配套光盘中的代码，相信您一读便懂。